Proceedings of the 6TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre
Banff, Alberta, Canada
October 6-11, 1991

In Cooperation with the
American Association for
Artificial Intelligence

Workshop Co-chairs:
John H. Boose, Boeing Computer Services
Brian R. Gaines, University of Calgary

Proceedings of the
6TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre, Banff, Alberta, Canada
October 6-11, 1991

In Cooperation with the
American Association for Artificial Intelligence

Workshop Co-Chairs:
John H. Boose, Boeing Advanced Technology Center
Brian R. Gaines, University of Calgary

These proceedings are being reissued for historical research purposes. Many of the workshop papers were further developed as journal publications and these constitute the primary scholarly record of the research.

6TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence
Banff, Canada, October 6-11, 1991

CONTENTS

Program iv

List of Participants ix

Technical Papers (listed alphabetically by first author)

KARL: an executable language for the conceptual model 1-1
Jürgen Angele, Dieter Fensel, Dieter Landes, Rudi Studer, University of Karlsruhe

KASE project toward effective diagnosis system developments 2-1
Dai Araki, Shoichi Kojima, Toshiba Corporation

Some theoretical issues on the computer integrated documentation project 3-1
Guy Boy, NASA-Ames Research Center

Knowledge representation for knowledge acquisition: a three schemata approach 4-1
Jeffrey M Bradshaw (Boeing Computer Services), Kenneth M. Ford (University of West Florida), Jack Adams-Webber (Brock University)

Case-based knowledge acquisition in the preformal phase 5-1
Sonja Branskat, University of Calgary

Ripple down rules: possibilities and limitations 6-1
Paul Compton, G. Edwards, B. Kang, R. Malor, T. Menzies, P. Preston, A. Srinivasan, S. Sammut, University of New South Wales & St Vincent's Hospital

Induction and visualization of rules with exceptions 7-1
Brian Gaines, University of Calgary

The automated knowledge engineer: natural language knowledge acquisition for expert systems 8-1
Yong Gao, Sharon Salveter, Boston University

The acquisition of syntactic knowledge from sample sentences 9-1
Fernando Gomez, University of Central Florida

An algebraic method for inducing efficient decision trees 10-1
Daniel Jaliff, University of Calgary

Experience in applying KRS to an actual business problem 11-1
Rob C. Kremer, Shell Canada Ltd

Creative knowledge acquisition: an analysis 12-1
Timothy C. Lethbridge, University of Ottawa

Using KADS to build ONCOCIN: a conceptual model of ONCOCIN 13-1
Marc Linster (GMD), Mark A. Musen (Stanford University)

More data on usable and reusable programming constructs 14-1
David Marques, Georg Klinker, Geoffrey Dallemagne, Patrice Gauthier, John McDermott, David Tung, Digital Equipment Corp

Towards an analysis and classification of approaches to knowledge acquisition from examination of textual metaphor 15-1
Mary A. Meyer (Los Alamos National Laboratory), Ray C. Paton (University of Liverpool)

Psychotherapeutic techniques for identifying experts' and system users' sensorial experience of thinking: significance to knowledge acquisition 16-1
Mary A. Meyer, Jeffrey M. Ross, Los Alamos National Laboratory

Extracting logical knowledge from prescriptive texts in order to build deontic knowledge bases 17-1
Bernard Moulin, Daniel Rousseau, Université Laval

Acquisition of knowledge for sharing and reuse 18-1
Robert Neches, ISI, University of Southern California

Knowledge adaptation: a means of knowledge acquisition 19-1
Alvaro de la Ossa, University of Kaiserslautern

A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools 20-1
Angel Puerta, John Egar, Sampson Tu, Mark Musen, Stanford University

A theory of local and interstitial knowledge 21-1
Alain T. Rappaport, Neuron Data

On the limitations of repertory grids in knowledge acquisition 22-1
Gordon Rugg, Nigel Shadbolt, University of Nottingham

On the acquisition and representation of procedural knowledge 23-1
Tim Saito (Computer Sciences Corporation), Chris Ortiz (NASA/Johnson Space Center), R. Bowen Loftin (University of Houston)

Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations 24-1
Franz Schmalhofer, Ralph Bergmann, Otto Kühn, Gabriele Schmidt, German Research Center for Artificial Intelligence

Studies in KADS interpretation models: formalization linking to domain layers and tool support 25-1
Wolfram Schmidt, Thomas Wetter, IBM Germany Scientific Centre

Knowledge acquisition for temporal abstraction mechanisms 26-1
Yuval Shahar, Sampson W. Tu, Mark A. Musen, Stanford University

Using knowledge acquisition tools to support creative processes 27-1
Mildred L. G. Shaw, Brian R. Gaines, University of Calgary

Expanding the descriptive power of guided self-elicitation 28-1
Glenn G. Shephard, San Jose State University

Terminology in knowledge acquisition: exploring a symbiotic relationship 29-1
Douglas Skuce, Ingrid Meyer, University of Ottawa

Knowledge acquisition for case-based consulting 30-1
Brian M. Slator, Kerim C. Fidel, Northwestern University

Generic techniques in EMA: a model-based approach for knowledge acquisition 31-1
Susan Spirgi, Dieter Wenger (Swiss Bank Corporation), André Probst (IBM Switzerland)

MULTIS: a knowledge acquisition system based on problem-solving primitives 32-1
Yuri Adrian Tijerino, Tadahiro Kitahashi, Riichiro Mizoguchi, Osaka University

Consensus knowledge acquisition 33-1
Andrew Trice (University of British Columbia), Randall Davis (MIT)

A problem-solving model for episodic skeletal plan refinement 34-1
Samson Tu, Yuval Shahar, John Dawes, James Winkles, Angel Puerta, Mark Musen, Stanford University

Acquiring first-order knowledge about air traffic control 35-1
Christel Vrain (Université Orléans), Yves Kodratoff (CNRS & Université Paris Sud)

Einstein—an interactive inductive knowledge acquisition tool 36-1
Geoffrey I. Webb, Deakin University

A framework for knowledge acquisition and machine learning 37-1
David C. Wilkins, University of Illinois

A comparison of structured vs. "unstructured" interviews for knowledge-acquisition 38-1
Larry E. Wood, Thomas C. Davis, Samuel L. Clay, John M. Ford, Brigham Young University

Developing K-ONCOCIN: a case study in the cognitive process of knowledge engineers 39-1
J. Brian Woodward, University of Calgary

PROGRAM

6TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

In Cooperation with the American Association for Artificial Intelligence
Banff, Alberta, Canada, October 6-11, 1991

Co-Chairmen
John H. Boose, Boeing Advanced Technology Center
Brian R. Gaines, University of Calgary

International Program Committee

Tom Addis, University of Reading, UK
John Boose Boeing Computer Services, USA
Guy Boy, Centre d'Etudes et de Recherches de Toulouse, France & NASA AMES, USA
Jeffrey Bradshaw, Boeing Computer Services, USA
B. Chandrasekaran, Ohio State University, USA
William Clancey, Institute for Research on Learning, CA, USA
Brian Gaines, University of Calgary, Canada
Jean-Gabriel Ganascia, University Pierre et Marie Curie, France
Thomas Gruber, Stanford University, USA
Koichi Hori, University of Tokyo, Japan
Nancy Johnson, Brunel University, UK
Georg Klinker, Digital Equipment Corp., USA
Shigenobu Kobayashi, Tokyo Institute of Technology, Japan
Yves Kodratoff, CNRS, France, & George Mason University, USA
Jae K. Lee, Korea Advanced Institute of Science and Technology, Korea
Marc Linster, GMD. Bonn, Germany
John McDermott, Digital Equipment Corporation, USA
Ryszard Michalski, George Mason University, USA
Riichiro Mizoguchi, Osaka University, Japan
Katharina Morik, GMD, Bonn, Germany
Hiroshi Motoda, Hitachi Ltd., Japan
Mark Musen, Stanford University, USA
Toyoaki Nishida, Kyoto University, Japan
Bruce Porter, University of Texas at Austin, USA
Ross Quinlan, Sydney University, Australia
Alain Rappaport, Neuron Data, USA
Mildred Shaw, University of Calgary, Canada
Hirokazu Taki, Mitsubishi Electric Corporation, Japan
Masanobu Watanabe, NEC Corporation, Japan
Bob Wielinga, University of Amsterdam, Holland

DAILY SCHEDULE

Workshop sessions will be held in the Bougeau Lounge on Floor 6 of the Lloyd Hall building. Continental breakfasts will be available before each session in the workshop area at 7:30. Sessions will start each day at 9:00. There will be coffee and juice breaks each morning and afternoon, and lunch will be served from 12:00 - 1:00, also in the workshop area. Hot breakfasts will be available in the cafeteria in the Donald Cameron building (main residential building). There will be group dinners Monday, Tuesday, and Thursday evenings in a dining room reserved for KAW'91 in the Donald Cameron building.

This meeting is a workshop, not a conference, and the program is targeted on promoting discussion on strategic and technical issues of importance to the KA community. Session speakers should use their 15 minute slot to address the issues of the session in relation to their own work, and not attempt to give a detailed research paper. Most of the time is scheduled for group discussion—from past workshops, it is never enough!

SCHEDULE OF EVENTS

Sunday, October 6

3:00 - 6:00 Registration; muffins & tea, bear jerky
7:00 pm Reception: Welcome & announcements, nuts, punch sharks, bears

Monday, October 7

Overview
9:00 - 9:40 Workshop Welcome and Introduction - Boose & Gaines

Reconstructing the Reconstruction of ONCOCIN—What Happens in Knowledge Modeling?
9:40 - 10:20 An overview of the ONCOCIN project
Mark A. Musen, Stanford University
9:20 - 10:40 Using KADS to build ONCOCIN: a conceptual model of ONCOCIN
Marc Linster (GMD), Mark A. Musen (Stanford University)

10:20 - 10:40 Break

10:40 - 11:00 Developing K-ONCOCIN: a case study in the cognitive process of knowledge engineers
J. Brian Woodward, University of Calgary
11:00 - 12:00 Discussion led by Georg Klinker, Digital Equipment Corp

12:00 - 1:00 Lunch

KADS Developments
1:00 - 1:20 Studies in KADS interpretation models: formalization linking to domain layers and tool support
Thomas Wetter, IBM Germany Scientific Centre
1:20 - 1:40 KARL: an executable language for the conceptual model
Rudi Studer, University of Karlsruhe
1:40 - 2:20 Discussion led by Franz Schmalhofer, German Research Center for Artificial Intelligence

2:20 - 2:40 Break

Origins of Knowledge in the Acquisition Process
2:40 - 3:00 Towards an analysis and classification of approaches to knowledge acquisition from examination of textual metaphor
Mary A. Meyer, Los Alamos National Laboratory
3:00 - 3:20 Creative knowledge acquisition: an analysis
Timothy C. Lethbridge, University of Ottawa
3:20 - 4:00 Discussion led by Nigel Shadbolt, University of Nottingham

4:00 - 6:00 Poster preparation time—for those who haven't put their posters up yet—pins, yarn, all kinds of goodies available

6:30 Dinner

Poster Reception
8:00 - 10:00 Nuts, punch and posters fresh off the press, videos, demos, interesting people

Tuesday, October 8

Strategic Issues—Knowledge Interchange—Project Sisyphus

9:00 - 9:40 Project Sisyphus—Activities to date
Marc Linster & Brian Woodward

9:40 - 10:20 Acquisition of knowledge for sharing and reuse
Robert Neches, ISI, University of Southern California

10:20 - 10:40 Break

10:40 - 11:00 Representations, standards, inter-operability and collaboration
Jeff Bradshaw, Boeing Computer Services

11:00 - 12:00 Panel and discussion on future directions for Sisyphus
Panel: Paul Compton, Marc Linster, Robert Neches & Brian Woodward
Discussion led by John Boose

12:00 - 1:00 Lunch

Dealing with Data

1:00 - 1:20 Consensus knowledge acquisition
Andrew Trice, University of British Columbia

1:20 - 1:40 Knowledge acquisition for case-based consulting
Brian M. Slator, Northwestern University

1:40 - 2:20 Discussion led by Doug Skuce

2:20 - 2:40 Break

Capturing Problem Solving

2:40 - 3:00 Knowledge adaptation: a means of knowledge acquisition
Alvaro de la Ossa, University of Kaiserslautern

3:00 - 3:20 KASE project toward effective diagnosis system developments
Dai Araki, Toshiba Corporation

3:20 - 4:00 Discussion led by Bernard Moulin

6:30 Dinner

Wednesday, October 9

Using Knowledge Acquisition Tools in Different Application Areas

9:00 - 9:40 Using knowledge acquisition tools to support creative processes
Mildred L. G. Shaw, University of Calgary

9:40 - 10:20 Some theoretical issues on the computer integrated documentation project
Guy Boy, NASA-Ames Research Center

9:40 - 10:20 Discussion led by Fernando Gomez

10:20 - 10:40 Break

Panel: Machine Learning and Knowledge Acquisition

10:40 - 12:00 Paul Compton, University of New South Wales
Christel Vrain, Université Orléans
Geoffrey I. Webb, Deakin University
David C. Wilkins, University of Illinois

Lunch, afternoon, evening - Local Atmosphere Checkup and Bear Patrol

Thursday, October 10

Stanford Medical Research Center Developments

9:00 - 9:15 A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools
Angel Puerta, Stanford University

9:15 - 9:30 A problem-solving model for episodic skeletal plan refinement
Samson Tu, Stanford University

9:30 - 9:45 Knowledge acquisition for temporal abstraction mechanisms
Yuval Shahar, Stanford University

9:45 - 10:20 Discussion led by Mark Musen

10:20 - 10:40 Break

Knowledge Primitives

10:40 - 11:20 A theory of local and interstitial knowledge
Alain T. Rappaport, Neuron Data

11:20 - 12:00 MULTIS: a knowledge acquisition system based on problem-solving primitives
Yuri Adrian Tijerino, Osaka University
Discussion led by Dieter Wenger

12:00 - 1:00 Lunch

Working Groups

Suggested Working Groups

Afternoon

Machine Learning (reporting: Christel Vrain, David Wilkins)
Knowledge Level Integration (Reporting: Jeff Bradshaw, Bob Neches)
Text Analysis (Reporting: Guy Boy, Fernando Gomez)
Generic Tasks (Reporting: David Marques, Franz Schmalhofer,)
Knowledge Acquisition: The Next 10 Years (reporting: John Boose, Dai Araki)

2:20 - 2:40 Break

6:30 Banquet Dinner with Annual Bear Tossing and Shark Snatching Mystery Guest Speaker

Friday, October 11

Working Group Presentations

9:00 - 9:30 Working Group 1
9:30 - 10:00 Working Group 2
10:00 - 10:30 Working Group 3

10:30 - 10:45 Break

10:45 - 11:15 Working Group 4
11:15 - 11:45 Working Group 5

11:45 - 12:15 Synthesis and Summary - Brian Gaines

12:15 - 1:00 Farewell Lunch

POSTERS

Knowledge representation for knowledge acquisition: a three schemata approach
Jeffrey M Bradshaw, Boeing Computer Services
Case-based knowledge acquisition in the preformal phase
Sonja Branskat, University of Calgary
Ripple down rules: possibilities and limitations
Paul Compton, University of New South Wales
Induction and visualization of rules with exceptions
Brian Gaines, University of Calgary
The automated knowledge engineer: natural language knowledge acquisition for expert systems
Yong Gao, Boston University
The acquisition of syntactic knowledge from sample sentences
Fernado Gomez, University of Central Florida
An algebraic method for inducing efficient decision trees
Daniel Jaliff, University of Calgary
Experience in applying KRS to an actual business problem
Rob C. Kremer, Shell Canada Ltd
More data on usable and reusable programming constructs
David Marques, Georg Klinker, Digital Equipment Corp
Psychotherapeutic techniques for identifying experts' and system users' sensorial experience of thinking: significance to knowledge acquisition
Mary A. Meyer, Los Alamos National Laboratory
Extracting logical knowledge from prescriptive texts in order to build deontic knowledge bases
Bernard Moulin, Université Laval
On the limitations of repertory grids in knowledge acquisition
Nigel Shadbolt, University of Nottingham
On the acquisition and representation of procedural knowledge
Tim Saito, Computer Sciences Corporation
Using integrated knowledge acquisition to prepare sophisticated expert plans for their re-use in novel situations
Franz Schmalhofer, German Research Center for Artificial Intelligence
Expanding the descriptive power of guided self-elicitation
Glenn G. Shephard, San Jose State University
Terminology in knowledge acquisition: exploring a symbiotic relationship
Douglas Skuce, University of Ottawa
Generic techniques in EMA: a model-based approach for knowledge acquisition
Susan Spirgi, Dieter Wenger, Swiss Bank Corporation
Acquiring first-order knowledge about air traffic control
Christel Vrain, Université Orléans
Einstein—an interactive inductive knowledge acquisition tool
Geoffrey I. Webb, Deakin University
A comparison of structured vs. "unstructured" interviews for knowledge-acquisition
Larry E. Wood, Brigham Young University

PARTIAL LIST OF DEMONSTRATIONS

Computer integrated documentation project
Guy Boy, NASA-Ames Research Center
KRS on Sisyphus project
Brian Gaines, University of Calgary
Expanding the descriptive power of guided self-elicitation (+video)
Glenn G. Shephard, San Jose State University
CODE knowledge visualization and inference
Douglas Skuce, University of Ottawa
EMA on Sisyphus project
Susan Spirgi, Dieter Wenger, Swiss Bank Corporation
Einstein—an interactive inductive knowledge acquisition tool
Geoffrey I. Webb, Deakin University
Cognosys on Sisyphus project
Brian Woodward, University of Calgary

LIST OF PARTICIPANTS

6th BANFF
KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff, Canada, October 6-11, 1991

Dai Araki
Systems and Software Engineering Laboratory
Toshiba Corporation
70, Yanagi-cho, Saiwai-ku
Kawasaki, 210 Japan

John H. Boose
Boeing Advanced Technology Center, 7L-64
Boeing Computer Services
PO Box 24346
Seattle, WA 98124 USA

Guy A. Boy
AI Research Branch
NASA-Ames Research Center
MS 244-17
Moffett Field, CA 94035 USA

Jeffrey M. Bradshaw
Boeing Advanced Technology Center, 7L-64
Boeing Computer Services
PO Box 24346
Seattle, WA 98124 USA

Sonja Branskat
Knowledge Science Institute
University of Calgary
Calgary, Alberta
Canada T2N 1N4

Paul Compton
School of Computer Science & Engineering
University of New South Wales
PO Box 1, Kensington
NSW 2010, Australia

Brian Gaines
Knowledge Science Institute
University of Calgary
Calgary, Alberta
Canada T2N 1N4

Yong Gao
Department of Computer Science
Boston University
111 Cummington St.
Boston
MA 02215, USA

Fernando Gomez
Department of Computer Science
University of Central Florida
Orlando, Florida 32751, USA

Daniel Jaliff
Knowledge Science Institute
University of Calgary
Calgary, Alberta
Canada T2N 1N4

Georg Klinker
Digital Equipment Corp
111 Locke Dr
Box 1006
Marlbouough, MA 01752 USA

Rob Kremer
Computing Research
Shell Canada Ltd
PO Box 100 Station M
Calgary, Alberta
Canada

Timothy Lethbridge
Dept of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5

Marc Linster
GMD
Postfach 1240
Scholß Birlinghoven
D-5205 Sankt Augustin 1
West Germany

David Marques
Digital Equipment Corp
111 Locke Dr
Box 1006
Marlbouough, MA 01752 USA

Mary A. Meyer
MS F600
Statistics Group
Los Alamos National Laboratory
Los Alamos, NM 87545 USA

Bernard Moulin
Département d'Informatique
Université Laval
Ste-Foy, Québec
Canada G1K 7P4

Mark Musen
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
School of Medicine
Stanford, CA 94305-5479, USA

Robert Neches
USC/ISI
4676 Admiralty Way
Marina del Rey
CA 90292, USA

Alvaro de la Ossa
Department of Computer Science
University of Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern, Germany

Angel Puerta
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
School of Medicine
Stanford, CA 94305-5479, USA

Alain Rappaport
Neuron Data
156 University Avenue
Palo Alto, CA 94301, USA

Tim Saito
Computer Science Corporation
16511 Space Center Boulevard - M30
Houston, Texas 77058, USA

Franz Schmalhofer
German Research Center for
Artificial Intelligence
University Bldg 57
Postfach 2080
Erwin-Schrödinger Str.
D-6750 Kaiserslautern FRG

Yuval Shahar
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
School of Medicine
Stanford, CA 94305-5479, USA

Nigel R. Shadbolt
Department of Psychology
University of Nottingham
Nottingham NG7 2RD, UK

Mildred L. G. Shaw
Knowledge Science Institute
University of Calgary
Calgary, Alberta
Canada T2N 1N4

Glenn G Shephard
College of Business
San Jose State University
One Washington Square
San Jose, CA 95192-0069, USA

Douglas Skuce
Dept of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5

Brian Slator
Institute for the Learning Sciences
Northwestern University
Evanston, IL 60201, USA

Susan Spirgi
Swiss Bank Corporation
Hochstrasse 16
4053 Basel, Switzerland

Rudi Studer
Institut für Angewandte Informatik
Universität Karlsruhe
PO Box 6980
7500 Karlsruhe Germany

Yuri Adrian Tijerino
Institute of Scientific and
Industrial Research
Osaka University
8-1 Mihogaoka, Ibaraki-shi
Osaka 567, Japan

Andrew Trice
Faculty of Commerce and
Business Administration
University of British Columbia
2053 Main Mall
Vancouver, B.C.
Canada V6T 1Z2

Samson Tu
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
School of Medicine
Stanford, CA 94305-5479, USA

Christel Vrain
LIFO, rue de Chartres
Université de Orléans
45067 Orléans cedex, France
91405, ORSAY, Cedex, France

Geoffrey I. Webb
Department of Computing and
Mathematics
Deakin University
Geelong 3217, Australia

Dieter Wenger
Swiss Bank Corporation
Hochstrasse 16
4053 Basel, Switzerland

Thomas Wetter
IBM Germany, Scientific Center
Institute for Knowledge-Based
Systems
Wilckensstr. 1a
D-6900, Heidelberg, Germany

David C. Wilkins
Department of Computer Science
University of Illinois at Urbana-
Champaign
Urbana, IL 61801, USA

Larry E Wood
Department of Psychology
Brigham Young University
Provo, UT 84602-1130, USA

Brian Woodward
Knowledge Science Institute
University of Calgary
Calgary, Alberta
Canada T2N 1N4

KARL: An Executable Language for the Conceptual Model

Jürgen Angele, Dieter Fensel, Dieter Landes, and Rudi Studer
Institut für Angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe, P.O. Box 6980, 7500 Karlsruhe, Germany
e-mail: fensel@aifb.uni-karlsruhe.de

Abstract

In this paper we give a detailed description of the "Knowledge Acquisition and Representation Language" (KARL), which is based on the main principles of KADS and allows a declarative and executable specification of the conceptual model. KARL consists of three well integrated parts: a data definition language for the domain layer, a logical language for the domain and inference layer, and a control flow language for the task layer. By providing an executable language for the conceptual model means are offered for integrating explorative prototyping into the life cycle oriented approach proposed by KADS.

1. MODEL-BASED KNOWLEDGE ACQUISITION

Knowledge Acquisition is a very critical phase in the process of building a knowledge-based system (kbs). In most cases, knowledge does not exist in a strictly formal manner which would allow direct input into a knowledge base. One possible way to deal with this problem is the **rapid prototyping** approach, compare e.g. [HWL83]. The evaluation of a quickly realised prototype guides knowledge acquisition. However this approch has some well-known disadvantages:

- There is no separation of knowledge level and symbol level [NEW82]. Problems of **what** the program should do are mixed with problems **how** this is realised within a specific knowledge representation and interpretation system.

- There is no objective guide for the knowledge acquisition process. Thus there is neither help to decide whether there is a lack of knowledge or not, nor support to determine when the gathering process should actually be finished.

- There is no explicit model of the expertise and the knowledge is only represented in a language which is not the experts´ language. Maintaining a kbs is analogous to the maintenance of a large traditional software program without any documentation.

Rapid prototyping is applicable if the kbs is used for academic research. But this kind of "methodology" is far from appropriate for building large knowledge bases dealing with real-life problems and meant to be used over long periods of time.

Among several alternative approaches to overcome the problems of rapid prototyping, we prefer the suggestion made by the KADS-group [BWS87, BRW89, HJK89, WSG89, WIS90]. Neglecting all the problems, inaccuracies, and ambiguities of this approach, we see five important principles of KADS:

1. KADS introduces a life cycle oriented methodology. The phases are knowledge acquisition, design, and implementation.

2. KADS distinguishes analysis and design. Thus, KADS clearly separates knowledge acquisition - the specification at the knowledge level what the program should do - from design and implementation of the program - the specification at the symbol level.

3. KADS separates different kinds of knowledge and provides different epistemological primitives for each of these types of knowledge. The result of the analysis phase, the conceptual model, consists of four different kinds of knowledge. This reflects the difference between traditional and AI software. Normally, AI problems cannot be solved by applying general algorithmic solutions. Instead, domain-specific heuristics must be used which describe how the program can fulfill its task. So when dealing with AI problems more kinds of knowledge must be elicited than in traditional software engineering [BRW89]:

- The **domain layer** contains domain specific knowledge about concepts, their features and their relationships. It is similar to the so-called content knowledge [MUS89]. This knowledge is task-neutral in the sense that there are no means to control its use. At this layer the objects of the domain are described which the kbs should use.
- The **inference layer** contains knowledge about the used problem-solving method in a way similar to [CLA85]. This layer indicates which inferences are necessary within the problem-solving method. This knowledge is meta-knowledge because it describes the application of the domain knowledge for inferencing. The relationships at the domain layer determine which inferences can be drawn at all. The inference layer determines which role domain knowledge shall play within these inferences.

  It consists of two kinds of elements: meta classes and knowledge sources. Meta classes provide or collect input or output of knowledge sources. They specify the concepts or structures of the domain layer which are premises or conclusions of inferences. Inferences are described via knowledge sources. A meta class can be a conclusion of one and a premise of another knowledge source. An **inference structure** is a network which shows these dependencies between meta classes and knowledge sources.
- The **task layer** contains knowledge about the control flow of a problem-solving method in order to solve a specific task. This layer specifies when inferences are made. Experiences with XCON show that control knowledge should be made explicit [SBJ87, BAC88].
- The **strategic layer** contains meta-knowledge about combination and selection of tasks in order to solve a problem adequately. Future work must be done to determine this layer more exactly.

4. KADS offers the chance for reusability of knowledge by clearly separating domain specific knowledge, task specific knowledge, and knowledge about the problem-solving method. The problem-solving method is independent of specific domains or tasks. This

is the foundation for reusability of program constructs. KADS proposes a library of generic problem-solving methods[1] which are called interpretation models.

5. Besides the advantages of reusability this library of interpretation models can guide knowledge acquisition. After a knowledge elicitation process which decides about which problem-solving method shall be applied, the selected model can be used as a template which must be filled with domain knowledge.

Apart from all its innovations, KADS has one big shortcoming. The result of the analysis phase, the conceptual model, is not executable. In traditional software engineering, the idea of executable specifications tries to combine the advantages of structured development methods and prototyping. This allows to evaluate the requirements with a running - though inefficient - system and avoids the normal disadvantages of prototyping. Because it is a kind of throw-away prototyping, there is no need to think about design and implementation of the real system and the evolving system is not an unstructured result of some prototype generations.

In fact, only by making the conceptual model executable two important requirements for efficient knowledge acquisition may be integrated into the KADS methodology:

1. The knowledge acquisition process is incremental and error-prone (see [MOR87] who calls it sloppy modeling). Therefore tools are needed which support the evaluation of acquired knowledge. All this is only possible with an executable specification.

2. During the last years, a lot of work has been done in the area of machine learning to improve the knowledge acquisition process. For example, inductive methods build decision trees [QUI90], conceptual descriptions [STM86, GLF90], modify logic derivations [MKK86], or create logical terms which describe domain and inference knowledge [MOR87, KOT87, MMS90]. In order to support this kind of automatic knowledge acquisition tools, formalisms are needed which can be directly used by computers.

In the next chapter , we will give an overview of the "Knowledge Acquisition and Representation Language" (KARL), which is a proposal for an operational specification based on the central ideas of KADS.

2. THE "KNOWLEDGE ACQUISITION AND REPRESENTATION LANGUAGE" (KARL)

2.1. General Remarks

First of all we want to make clear that KARL does not aim at being a language for describing the final kbs in the design or implementation phase. A lot of effort has been spent in database technology in order to develop semantic data models. Similarly, in AI much research has been done on knowledge representation formalisms: semantic networks in systems like KL-ONE [BRS85], NIKL [MOS83], SB-ONE [KAL86], BACK [LNP87] and frames in systems like KEE [FIK85]. In contrast to knowledge representation formalisms semantic data models are not executable and only define an intermediate step in the design of data base systems. Our claim is neither to invent a new knowledge representation formalism nor a new variant of semantic data model.

---

[1] [CHA86] also suggests reusable program constructs and calls them generic tasks. He differs from our proposal because he suggests different representation languages for different problem types.

One central idea of KADS is the separation of analysis and implementation. We want to make the conceptual model executable in order to allow explorative or throw-away prototyping [FLO84] during the analysis phase. KARL is not meant to be used as a language for implementing the final kbs. Consider software engineering as an analogy: we do not want to create a new programming language; instead we would like to have a specification language like PSL [TEH77] or RSL [BBD77] to allow executable specifications.

Second the language as presented here is just the core of the language. If we realize that we need additional features (e.g. multiple inheritance) for effective prototyping these will be integrated in future versions of KARL.

KARL consists of three parts:

- A data definiton language which allows to define relation types at the domain layer. These relation types are arranged in a hierarchy of superrelation types and subrelation types. Any relation type may be further characterised by means of attributes attached to it.

- a logical language which is used at the domain and inference layer. At the inference layer it serves for describing how generic knowledge is related to specific knowledge at the domain layer and which inferences are possible. At the domain layer the logical language is used to express how elements of relations are related and which constraints must be met.The logical language allows to build formulae from predicates (interpreted by relations) which are combined by means of logical operators, namely "and" (&), "or" (|), "implication" (<-), and "negation" (-). Negation has to be "stratified" [Ull89] thus restricting its use. It is important to note, however, that no functions are allowed. We restrict the set of formulae to those that do not transgress the expressive power of Horn logic plus stratified negation. In KARL, the premises of clauses are literals which may be combined using the logical operators mentioned above. The conclusions must be single positive literals.

- a language which describes at the task layer **when** and **which** inferences are to be drawn. This language is used to describe control flow. It contains constructs for subtasks (procedures), recursion, sequence, alternative, and iteration of calls to knowledge sources.

An impression of how these languages are used for modelling may be gained from [AFL91b].

2.2. The Different Layers

2.2.1. The Domain Layer

In contrast to KADS we divide the domain layer into two parts: a *knowledge part* and a *data part*. The *knowledge part* contains the knowledge modeled by the knowledge engineer. This knowledge is static while the kbs is in action i.e. it does not change while the kbs handles actual cases. The other part of the domain layer, the *data part*, contains case specific data. The data part is intialized with input data before the problem-solving starts and it will contain the result of the system when the problem-solving process has finished. Intermediate data may be written to and read from this part. Thus the problem-solving process may read from both parts of the domain layer but may only write to the *data part*. Both parts are described by the same modeling primitives which are presented in the following.

The static knowledge needed for the problem-solving capabilities of the system is represented at the domain layer. It is used to describe objects and their relationships which the expert system may use for problem-solving. In KARL the modeling primitives to describe them are ***relation type, instance, intensional description, "is-a" hierarchy*** and

constraints. These primitives can be used to describe ***concepts, relationships***, and ***facts*** which hold in a domain.

A ***relation type definition*** is an expression:
"$R(t_1:R_1, ..., t_n:R_n)<a_1:\chi_1, ..., a_m:\chi_m>$"
"R" is called the ***name*** of the relation type. Each "$t_i:R_i$" is called a ***component*** of the relation type, where "t_i" is the ***name of the component***, and "R_i" is the ***type name of the component***. Any type name of a component must either be equal to "All" or must also be defined as a name of a relation type. Each "$a_j:\chi_j$" is called an ***attribute*** of the relation type, where "a_j" is the ***name of the attribute***, and "χ_j" is the ***type name of the attribute***. Presently only the type which offers all alphanumerical strings with finite length exists. To allow quick and easy prototyping we did not introduce a more sophisticated type concept for attributes.

Let $O := \{o_1,...,o_k\}$ be a finite set of objects. An ***instance*** of the relation type "R" is a tuple of objects and attribute values. The structure and the arity of the tuple depends on the components of "R". The i-th component of an instance of "R" must be an instance of "R_i" for all i=1,..,n.

An ***intensional description*** of a relation type is possible with KARL clauses, e.g. by

$R_1(t_1:X) <- R_2(t_2:X) \& R_3(t_3:X).$

$R_1(X)<a:Y> <- R_2(X)<b:Y>.$

As relation types have named components and attributes, it is not necessary to mention all of these in a KARL clause. Only the relevant components and attributes must be listed.

Any relation type can be an element of an ***"is-a" hierarchy***. The "is-a" hierarchy introduces sub and super relation types in combination with attribute inheritance among them. Therefore for a subrelation type only the additional attributes must be defined in its definition. We write

$R_1 < R_2$

if the relation type "R_1" is a subrelation type of the relation type "R_2". In this case "R_1" inherits all attributes of "R_2" and every instance of "R_1" inherits all attributes values of the corresponding instance of "R_2".

In addition to the introduced primitives, ***constraints*** may be expressed with KARL clauses. With constraints we can express that, e.g., no object is both an instance of the relation type "republican" and an instance of the relation type "quaker":

<- quaker(person:X) & republican(person:X).

Constraints are rules about the structure of the domain and are not used for inferencing, but to ensure consistency. They specify necessary conditions for the systems behaviour. In KADS constraints have not yet been used.

These language primitives can be used to model regularities which hold in a domain. A ***concept*** is modelled by an unary relation type, a ***relationship*** is modelled by a relation type, both can be further described by intensional descriptions and constraints. ***Facts*** are described by instances of relation types.

For a comparision of the expressive power of the language used at the domain layer and the semantic data model SDM [BRR84] compare [AFL91a].

2.2.2. The Inference Layer

The inference layer is like a "wooden iron" in that it describes **generic** inferences which use **domain specific** knowledge. So we think that this twin role is the real problem of KADS and its operationalisation. These two kinds of knowledge must be separated clearly, i.e. for both kinds of knowledge, specific language elements with clearly separated functions must be offered:

- On the one hand, the knowledge source and the inference structure should only consist of **generic** knowledge.
- On the other hand, a view on domain knowledge must be established which allows the knowledge source to use this knowledge.

If this can be achieved, reusability is possible in two ways: First, the knowledge about the problem-solving method is domain and task independent, i.e it may be used in several domains and for different tasks. Second, the same domain knowledge may be used for different problem-solving methods and different, though related tasks (cf e.g. [KBD89]). Therefore, domain knowledge may be reused as well.[2]

In the following, we describe the language elements which KARL provides at the inference layer.

Meta classes

Similar to KADS, a meta class serves as a slot or container which can be filled with domain concepts or structures [BRW89]. Two types of meta classes, namely input meta classes and output meta classes, can be distinguished both of which contain a set of flat, i.e. unstructured, tuples. Input meta classes provide input to a knowledge source whereas results produced by a knowledge source are collected in an output meta class in case that the respective inference is drawn in a data-driven manner.

In traditional software engineering, e.g. in SADT [ROS77], it has been useful to distinguish data which are subject to a transformation within a module from data which merely influence the way the transformation is carried out. In a similar manner, [LIN90] proposes control meta classes as a third type of meta classes in order to distinguish knowledge that is actually transformed by a knowledge source (input meta classes) from knowledge that merely influences this transformation. This seems reasonable to us as it may improve the clarity of the conceptual model. We found, however, that it is often hard to determine if knowledge is to be described as input meta class or as control meta class. In heuristic classification [CLA85], for example, it is not clear if the abstracted data which are heuristically matched to solution classes serve as input or as control knowledge. Therefore we decided not to integrate control meta classes in KARL at the moment.

Meta classes serve as passive elements which are filled with knowledge. With some meta classes, however, an additional active role is associated by connecting them to corresponding domain knowledge. Details of how this connection, the mapping to the domain layer, is established are given below.

---

[2] In our opinion it is impossible to model domain knowledge without a task serving as guideline. However, for the success of kbs technology it must be possible to reuse knowledge as the cost for gathering it anew for each application is much too high.

Meta relations

Additionally, we need so-called meta relations. A knowledge source performs its inference by using certain relations. In the knowledge source, these relations are addressed as generic relations, i.e. as meta relations. In an analogous manner to meta classes, meta relations also have to be linked to the domain layer. Again, details about this linkage can be found below.

Knowledge sources

„A knowledge source is a functional description of an inference making process" [BRW89]. A knowledge source has one or more input meta classes, meta relations and output meta classes[3] (which can be used as input meta classes by other knowledge sources). A knowledge source rewrites the content of its output meta classes. A knowledge source can be described as follows:

```
knowledge source: classify
    begin
            input: imc.
            output: omc.
            metarelation: metarel.
            knowledge source:
                    omc(X) <- imc(Y) & metarel(Y, X).
    end
```

This means that the knowledge source *classify* takes its input from the input meta class *imc*, transforms these data by the meta relation *metarel*, and writes the results to the output meta class *omc* (in case of data driven evaluation).

Inference structure

For the solution of most problems, several knowledge sources must cooperate. This cooperation is established via meta classes, e.g. the output meta class of one knowledge source serves as input meta class for another knowledge source:

```
knowledge source: classify
    begin
            input: object_description.
            output: object_class.
            metarelation: mr_classify.
            knowledge source:
                    object_class(X) <- object_description(Y) &
                                                    mr_classify(Y, X).
    end

knowledge source: select
    begin
            input: object_description.
            input: object_class.
            output: solution.
            metarelation: mr_select.
```

---

[3] KADS allows only one output meta class for a knowledge source. We think that this restriction is not very useful. If e.g. one needs to select an element of a set and one also needs the remaining set, two knowledge sources with a single output meta class each would be needed in order to provide the respective items. This can be simplified if knowledge sources with more than one output meta classes are introduced.

knowledge source:
solution(Y) <- object_class(Y) &
-(object_description(Z) & object_class(X) &
mr_select(X, Y, Z)).
end

The clause within the description of the knowledge source *classify* states that in case of data-driven evaluation, all instances of the input meta class *object_description* which also appear as the first component of an instance of the meta relation *mr_classify*, are written to the output meta class *object_class*.

Similarly, *select* puts those instances of the input meta class *object_class* into the output meta class *solution* for which there are no instances of *object_description* and *object_class* such that *mr_select* also holds.

If knowledge sources shall cooperate, the identifiers of the meta classes must be adapted so that the same names are used by any knowledge source which refers to them. In the example above, the meta class *object_description* is used as input by both knowledge sources whereas the output meta class of *classify* serves as an additional input meta class for *select*. Corresponding names implicitly define the inference structure.

KADS proposes a library of predefined combinations of knowledge sources which describe generic problem-solving methods. These combinations are called interpretation models. Interpretation models can be combined in an analogous manner as elementary knowledge sources. The user must only adapt the names of those meta classes which link the interpretation models. It is also possible to connect elementary knowledge sources and interpretation models.

It should be noticed that no control knowledge is represented at the inference layer. Neither the sequence of definitions of knowledge sources, nor the sequence of clauses within the description of a knowledge source does imply an order of execution.

For easier handling of the complexity of an inference structure for complicated problems we introduce a hierarchical ordering of substructures like in SADT [ROS77] or in data flow diagrams [YOU89]. The highest level - the so called context diagram - offers a general overview of the entire problem-solving method and its input and output meta classes (see figure 1).

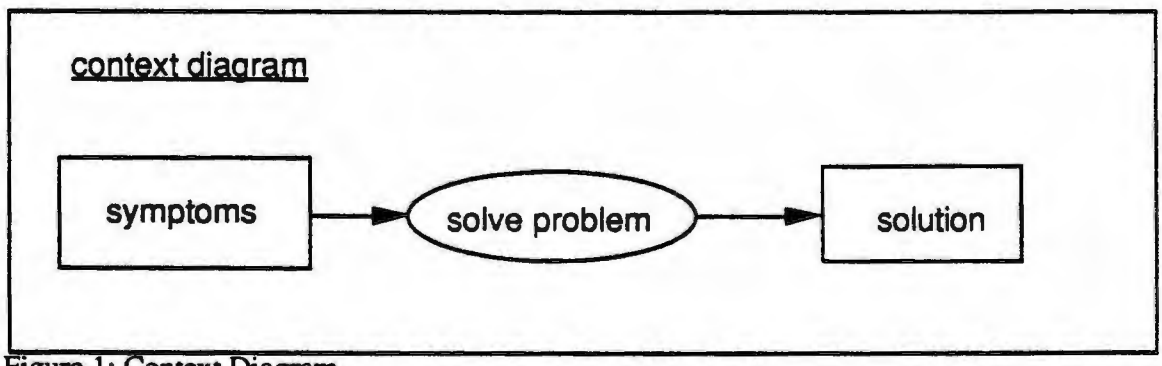


Figure 1: Context Diagram

In the following refinement step (see figure 2) the inference which represents the problem-solving method as a whole is decomposed into its parts, e.g. the next step may contain different interpretation models which interact to solve the problem. In further steps these interpretation models must be specified. This has to be repeated until elementary knowledge sources are reached. We also allow recursion, i.e. the refinement of a bubble, a so-called **inference model**, may e.g. contain the same bubble again. In this case the internal bubble

must have the same environment as the superior bubble, i.e. input and output meta classes must have the same names and meta classes and knowledge source have to be connected in the same way. An inference model may communicate with its environment by reading data from and writing data to meta classes of its superior inference model, i.e. the one which contains a bubble which is refined by the former inference model. Such meta classes are referred to as *external* in the more specific inference model.

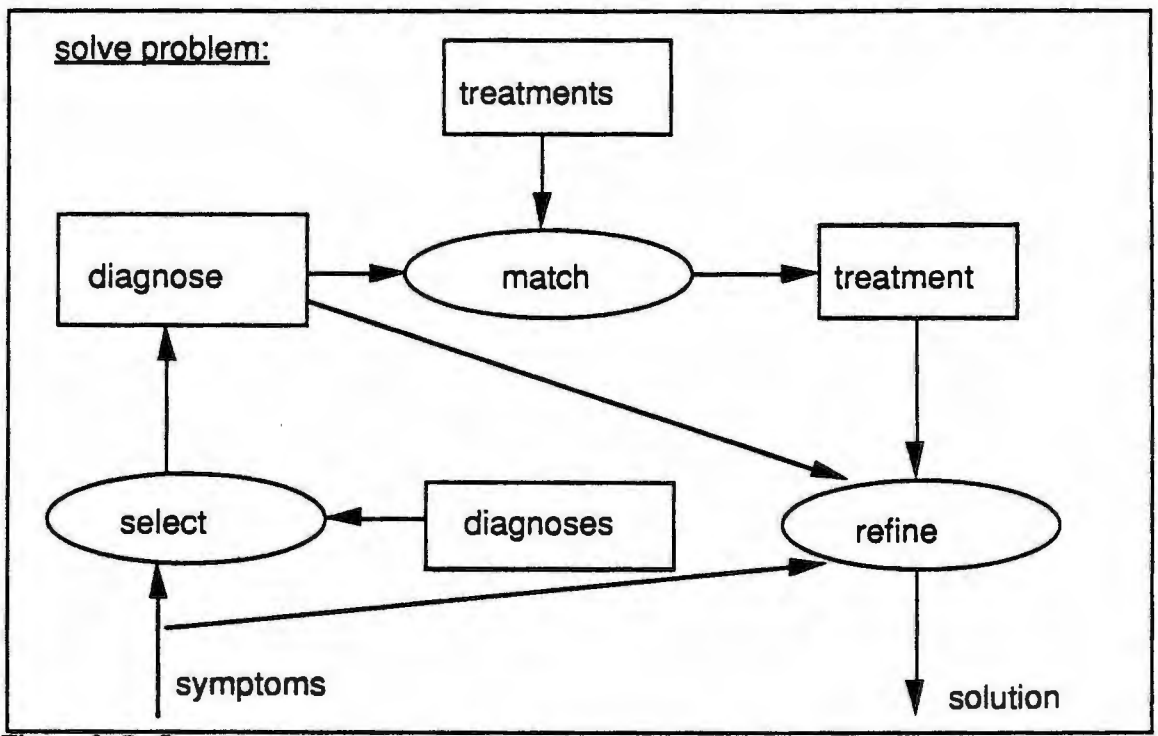


Figure 2: Refinement

The rules for the decomposition of inference structures are identical to those for well-formed data flow diagrams [YOU89], plus some rules which cover recursion. A tool to be developed will check the consistency of the different parts of the inference structure.

2.2.3. Task Layer

Experiences in XCON [SBJ87, BAC88] show that great problems arise if control flow is implicitly specified in the rules. Therefore control flow has been made explicit in the new implementation of XCON in RIME. In KADS, knowledge about control flow is also clearly separated from other kinds of knowledge by describing it at a separate layer of the conceptual model, namely the task layer [WSG89].

It is well-known that the control flow constructs **sequence, iteration**, and **alternative** are sufficient to describe any computable function. In KARL, sequence is indicated by the sequential arrangement of statements. The other control flow constructs at the task layer have the following form (the second type of iteration is included for convenience only):

if..then..else..endif	(alternative)
while..do..endwhile	(iteration)
repeat..until..	(iteration)

In some cases, problem-solving methods can be described more adequately in a recursive fashion. Therefore, we also allow **recursion** as a construct at the task layer. In future, we may also allow parallelism since subtasks may frequently be carried out concurrently.

As elementary statements, a program at the task layer, a so-called task structure, contains calls of knowledge sources. Calls to knowledge sources may be combined to named **subtasks** (cf e.g. [BWS87]), an analogon to procedures in programming languages. Only those calls to knowledge sources may be combined to subtasks which constitute an inference model within the inference structure at the inference layer since only such a combination of knowledge sources represents a meaningful way of solving a subproblem. The respective inference model implicitly defines the input and output meta classes of the subtask. Therefore, specifying input and output parameters of substructures at the task layer is omitted as it would be redundant. A knowledge source may be a member of no more than one subtask whereas subtasks may be used in various other subtasks.

Since inference models may be refined by other inference models, subtasks may be nested, too, including recursive calls. It should be noted that in case of recursion, the meta classes used by a subtask are duplicated, i.e. recursive calls work on local copies of the respective data. Only at the top level of recursion, the original meta classes are accessed. If data used in lower level calls are to be preserved, they may be written to the data part of the domain layer.

Both, elementary knowledge sources and subtasks, return boolean values as results in order to indicate if the call has been successful. In subtasks, *return* statements are used to exit to the calling program or subtask. As a parameter of the *return* statement, a boolean value to be returned to the calling subtask or task structure is given. Boolean return values can be used for selecting alternatives or stopping iterations at the task layer. Additionally, these return values can be stored in boolean variables which are another element of task structures or subtasks at the task layer. The scope of these variables is global, i.e. in contrast to meta classes, no duplication of variables takes place in case of recursive calls to subtasks.

The task layer also determines whether the evaluation is data-driven or goal-driven. If the evaluation should be data-driven the calls of the knowledge sources are in the same sequence as the direction of the inference indicated by the input meta classes and the output meta classes. If the evaluation should be goal-driven the sequence of the calls of the knowledge sources is reversed. In this case, contents of output meta classes are used as input for the knowledge sources and results are written to input meta classes.

2.2.4. The Strategic Layer

The strategic layer allows to use e.g. meta reasoning and machine learning. But so far there is no clear view how this might work and the role of the strategic layer in general is not yet defined exactly[4]. Therefore, we do not propose any executable language primitives for this layer.

[4] Currently work including this topic is carried out in the REFLECT project (Esprit project no. 3178)

2.3. The Connections of the Different Layers

2.3.1 The Connection between Inference and Domain Layer

For meta classes and meta relations their connection to the domain layer has to be specified.

Meta classes

Meta classes at the inference layer may be given an active role by connecting them to the domain layer. Knowledge at the domain layer serves primarily as input, but it is also possible to write results of knowledge sources back to the domain layer. Knowledge which is written to the domain layer is case specific, i.e. it can be distinguished from general domain knowledge which holds irrespective of the actual case at hand. This distinction is reflected by the fact that we distinguish two parts of the domain layer: a knowledge part, which contains general domain knowledge, and a data part, which contains case specific domain knowledge.

Meta classes which are not connected to the domain layer contain intermediate results which are of no interest at the domain layer.

The connection between the domain layer and an input meta class **transforms** specific domain knowledge about various concepts and relationships. Within the meta class, this transformed knowledge then serves as input for all the knowledge sources at the inference layer refering to this meta class as input meta class. The connection of the meta class to relation types at the domain layer is specified in a KARL clause. The premise of the clause contains literals addressing the respective domain relation types; the conclusion is a literal addressing the meta class, e.g.

meta_class(X) <- domain_relation(name$_1$: X, name$_2$: a).

This expression states that the relation type *domain_relation* describes the set of objects which a (generic) knowledge source refers to by its input meta class *meta_class*.

Similarly, the connection between an output meta class and the domain layer describes how knowledge which is inferred at the inference layer may be stored in a domain relation type. Again, this is specified with a KARL clause. The conclusion of the clause addresses the domain relation type and the premise indicates which meta class provides the information to be stored, e.g.

domain_relation(name:X) <attr: Z> <- meta_class(X, Z).

This expression states that the contents of an output meta class *meta_class* are stored at the domain layer, namely in the relation type *domain_relation*. The second argument of *meta_class* corresponds to an attribute of *domain_relation*.

Meta classes which are connected to the domain layer constitute a view of the problem-solving method onto the domain knowledge - compare the discussion to view definition in database design, e.g. [BLN86, ELN89]. Since a piece of domain knowledge may play various roles within a problem-solving method, several meta classes may be mapped onto the same relation type at the domain layer.

Meta relations

Similar to meta classes, meta relations allow to define **new** relationships which are not defined at the domain layer. Relation types which are defined at the domain layer are used for specifying meta relations. This idea is a consequence of our notion how to connect meta classes

to the domain layer. However, meta relations may not write information back to the domain layer.

A meta relation **transforms** domain specific relation types so that knowledge sources can use them, e.g.

meta_relation (X_1, X_3) <-
domain_relation$_1$(*name*$_1$: X_1, *name*$_2$: X_2) &
domain_relation$_2$(*name*$_3$: X_3, *name*$_4$: c).

This expression states that the generic relation *meta_relation* which a knowledge source uses for drawing inferences corresponds to the conjunction of the domain specific relation types *domain_relation*$_1$ and *domain_relation*$_2$.

2.3.2 The Connection between Inference and Task Layer

Programs at the task layer execute knowledge sources and subtasks. Upon execution, knowledge sources and subtasks return boolean values to the task layer which are used for the selection of alternatives or for controlling iterations. A knowledge source has been executed successfully if the contents of the meta classes it uses have been modified during its execution. The return value of a subtask depends on the return values of the knowledge sources and subtasks it contains.

3. COMPARISON WITH SIMILAR APPROACHES

Other proposals for executable languages for conceptual models are made by [AHS90, LIN90, KVS91, WET90].

[WET90] suggests a powerful but uniform representation formalism based on predicate logic. We think that a central idea of KADS, namely to distinguish different kinds of knowledge, makes it necessary to use different formalisms for different layers.

The most important difference of KARL compared to [LIN90] is the way the inference layer is handled, a central issue in operationalising KADS:

- We give an active role to the meta classes: they define a view of the generic knowledge sources onto specific domain knowledge. They are more than just passive collectors of instances.

- [LIN90] proposes an automatic code generator adding domain specific knowledge to the knowledge sources. With our view definition we can describe the knowledge sources in a generic way. We think that our declarative description language for knowledge sources gives us greater flexibility than an automatical code generator.

- Because the tuple-oriented approach of Prolog, as it is used in [LIN90], has some deficiencies we prefer to take a set-oriented approach for the evaluation of clauses in KARL. This approach is also used in several inference mechanisms of deductive data base systems.

[AHS90] propose the language $(ML)^2$ for describing conceptual models in a formal manner. A subset of this language is executable and thus allows prototyping in a similar way as our approach. In $(ML)^2$ the languages of task, inference and domain layer are based on a uniform formalism: predicate logic. For the specific needs of inference and task layer, conservative extensions of pure predicate logic, namely meta logic and dynamic logic, are used. For the

mapping between the inference and domain layer $(ML)^2$ provides a *lift*-operator. This mechanism seems to be slightly more general than the mapping used in KARL. In contrast to $(ML)^2$ KARL allows to use recursion as a control primitive at the task layer. With recursion the whole state of a problem solving process is accessible at one time. It is very difficult to reconstruct the whole state of the problem solving process with the history mechanism for each meta variable as it is implemented for $(ML)^2$. The knowledge expressed in $(ML)^2$ is rather difficult to read and understand and thus may hardly be used as medium of communication with experts as it is proposed in papers about $(ML)^2$.

Model-K [KVS91], another language which aims at making conceptual models of KADS executable, is built on top of the expert system shell BABYLON [CDV89]. In contrast to our approach Model-K has the claim to cover the entire life-cycle from the conceptual model to the final implementation of the expert system. These different stages of the life cycle correspond to different levels of detail of a model expressed in Model-K. Contrary to that, a conceptual model expressed in KARL serves as input for the design process, but is not successively extended to the final expert system.

Model-K does not allow to describe the expertise in the conceptual model in sufficient detail since only the rough task, inference and domain structures are described in this model. Later on control statements, knowledge source bodies, and concept instances have to be supplied for making the description executable thus bringing this model closer to implementation and taking it away away from "conceptual" model. Programming knowledge sources in lisp, prolog etc. prevents the expert from understanding what these knowledge sources do and how they work. We feel that explicit declarative descriptions of what knowledge sources should do and how they achieve this functionality are necessary in order to establish a basis for communication with experts. Therefore though the structure of the whole system is described at the knowledge level, the development of any single knowledge source is closer to prototyping at the symbol level.

4. FUTURE DEVELOPMENTS

Experience with KARL may show the need for ***more powerful language primitives***. So KARL eventually has to be extended, e.g. by introducing modularisation as a feature of the data definition language.

KARL has to be encapsulated by a sophisticated human interface, because we do not think the knowledge engineer express the knowledge directly in KARL. For instance elements of relations are edited by using formulas etc.

Tools have to be developed supporting the ***bottom up*** modeling process of ***translating an informal specification into a formal one*** expressed by KARL. These tools have to support the knowledge engineer to bridge the large gap between informal and formal descriptions in a semi-automatic process by using mediating representations as intermediate stages. ***Inductive tools*** construct, e.g., "is-a" hierarchies of the relation types, their schema definitions, and their intensional descriptions using instances of the relation types as an input [FEN91]. The ***top down*** approach to modelling has to be supported by matching process knowledge [MUS89] gathered in an early acquisition phase with already existing problem solving methods and by helping to use these problem solving methods as templates in order to model the domain layer.

There is a need for an environment supporting the ***validation*** of a model written in KARL. The ***execution*** of a model may be supported by e.g. a debugger. ***Symbolic execution*** means to apply the inferences at the inference layer to preconditions of the inference layer (e.g. constraints at the domain layer) in order to derive post conditions of the inferences. ***Verification*** means to derive preconditions derived from postconditions of the inference

layer and checked against the knowledge at the domain layer The domain layer has to be checked for ***redundancy*** and ***consistency*** [ANG91]. The model may be ***refined*** automatically if it exhibits an undesired behaviour (cmp. [GIN88]). Thus this environment supports the ***prototyping process*** with an already existing model written in KARL.

Third, if a system specification is written down in KARL the next step has to ***transform this specification into a design model*** and later on into an implementation. The closer the implemented system is to the specification, the easier the maintenance of the system will be. Further research must be done to investigate methods and tools supporting this transformation.

5. CONCLUSION

With the Knowledge Acquisition and Representation Language (KARL) it is possible to integrate the advantages of explorative prototyping and structured development methodologies like KADS to build kbs. We think that both is necessary to build large kbs for long time use concerned with real-life problems. We propose this on the background of the experience in the field of software engineering where one also tries to integrate life-cycle oriented methodologies with prototyping. With KARL as a foundation, sloppy modelling supported by learning and KADS are no longer in conflict. Both are one way to build a kbs.

Acknowledgement

We gratefully thank Angi Voß, Peter Sander, and Barbara Messing for helpful critique and discussions.

Bibliography

[AHS90] Akkermans, H.; van Harmelen, F.; Schreiber, G.; Wielinga, B.: A formalisation of knowledge-level models for knowledge acquisition. In *International Journal of Intelligent Systems*, 1990, Forthcoming

[AFL91a] J. Angele, D. Fensel, and D. Landes: Two Languages to Do the Same? In Research Report, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, University of Karlsruhe, 1991.

[AFL91b] J. Angele, D. Fensel, D. Landes, and R. Studer: Sisyphus - no problem with KARL. In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91*, Crieff, Scotland, May 20-24, 1991.

[ANG91] J. Angele: Checking knowledge bases for inconsistency and redundancy. In Research Report, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, University of Karlsruhe, no. 228, 1991.

[BAC88] Bachant, J.: RIME: premliminary work toward a knowledge-acquisition tool. In *Automating Knowledge Acquisition for Experts Systems*, S. Marcus, Kluwer Academic Publisher, Boston, 1988.

[BAR86] Bancilhon, F.; Ramakrishnan, R.: An amateur's introduction to recursive query processing strategies. In: *ACM SIGMOD*, 1986.

[BBD77] Bell, T.E.; Bixler, D.C.; and Dyer, M.E.: An extendable approach to computer-aided software requirements engineering. In *IEEE Transactions on software engineering*, vol. SE-3, no.1, January 1977.

[BLN86] Batini, C.; Lenzerini, M.; and Navathe, S.B.: A comparative analysis of methodologies for database schema integration. In *ACM Computing Surveys, vol. 18, no. 4*, Dec. 1986, 323-364.

[BRR84] M.L. Brodie and D. Ridjanovic: On the design and specification of database transactions. In M.L. Brodie et.al. (eds.), *On Conceptual modelling. Perspectives from Artificial Intelligence, databases, and Programming languages*, Springer-Verlag, New York, 1984.

[BRS85] Brachman, R.J. and Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. In: *Cognitive Science*, no. 9, 1985.

[BRW89] Breuker, J.; and Wielinga, B.: Models of expertise in knowledge acquisition. In *Topics in Expert Systems Design*, G. Guida and C. Tasso (eds.), Elsevier Science Publisher B. V., North-Holland, 1989.

[BWS87] Breuker, J.; Wielinga, B.; Someren, M.v.; de Hoog, R.; Schreiber, G.; de Greef, P.; Bredeweg, B.; Wielemaker, J.; and Billault, J.-P.: Model-driven Knowledge Acquisition: Interpretation Models. Esprit Project P1098, University of Amsterdam (The Netherlands), 1987.

[CDV89] Christaller, T.; Di Primio, F.; Voss, A.: *Die KI-Werkbank BABYLON*. Addison Wesley, Bonn, 1989.

[CHA86] Chandrasekaran, B.: Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. In: *IEEE Expert*, no. 3, vol. 1, 1986, 23-30.

[CHE76] Chen, P.P.: The entity relationship model. Toward a unified view of data. In *ACM Transactions on Database Systems*, vol. 1, no. 1, 1976, 9-36.

[CLA85] Clancey, W.J.: Heuristic classification. In *Artificial Intelligence 27*, 1985, 289-350.

[EKK89] Emde, W.; Keller, I.; Kietz, J.U.; Morik, K.; Thieme, S.; and Wrobel, S.: Wissenserwerb und maschinelles Lernen, Synthesis report of the KIT-LERNER project, KIT-Report 71, Technische Universität (TU) Berlin, 1989.

[ELN89] Elmasri, R.; Navathe, S.B.: *Fundamentals of Database Systems*, Benjamin Cummings, Redwood City, Cal., 1989.

[ESH88] Eshelman, L.: MOLE: a knowledge-acquisition tool for cover-and-differentiate systems. In *Automating Knowledge Acquisition for Experts Systems*, S. Marcus, Kluwer Academic Publisher, Boston, 1988.

[FEN91] Fensel, D.: Modeling of content knowledge with conceptual clustering. In Research Report, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, no. 219, University of Karlsruhe, 1991.

[FIK85] Fikes, R.; Kehler, T.: The role of frame-based representation in reasoning. In *Communications of the ACM 28*, No. 9, 1985, 904-920.

[FLO84] Floyd, C.: A systematic look at prototyping. In *Approaches to Prototyping*, R. Budee et. al. (eds.), Springer-Verlag, Berlin, 1984.

[GIN88] A. Ginsberg: Theory revision via prior operationalization. In: *Proceedings of the 7th National Conference on Artificial Intelligence*, 1988.

[GLF90] Gennari, J.H.; Langley, P.; and Fisher, D.: Models of incremental concept formation. In *Machine Learning. Paradigms and Methods*, J. Carbonell (ed), Elsevier Science Publisher B.V., Amsterdam, 1990.

[HWL83] Hayes-Roth, F.;Waterman, D.; and Lenat, D.: *Building Expert Systems*. Addision Wesley Publishing Company, London, 1983.

[HJK89] Hickman, F.R.; Killin, J.L.; Land, L.; Mulhall, T.; Porter, D.; and Taylor, R.M., *Analysis for Knowledge-Based Systems: a practical guide to the KADS methodology*, Ellis Horwood Limited, Chichester, GB, 1989.

[KAH88] Kahn, G.: MORE: from observing knowledge engineers to automating knowledge acquisition. In *Automating Knowledge Acquisition for Experts Systems*, S. Marcus, Kluwer Academic Publisher, Boston, 1988.

[KAL86] Kalmes, J: SB-Graph User Manual. Fachbereich Informatik, Universität Saarbrücken, 1986.

[KVS91] Karbach W, Voß A, Schuckey R, Drouven U.: MODEL-K: Prototyping at the knowledge level. In: Proceedings of the 11th International Conference on Expert Systems and their Applications, vol.1, General Conference Tools, Techniques and Methods (Avignon, France, May 27-31), 1991, pp 501-511

[KBD89] Klinker, G.; Bhola, C.; Dallemagne, G.; Marques, D.; and McDermott, J.: Usable and reusable programming constructs. In *Proc. of the 5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop* (Banff, Canada, Nov.), 1989.

[KOT87] Kodratoff, Y.; and Tecuci, G.: The central role of explanation in DISCIPLE. In *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed.), Springer Verlag, Berlin, 1987.

[LIN90] Linster, M.: Declarative problem-solving procedures as a basis for knowledge-acquisition: a first proposal. In *Arbeitspapiere der Gesellschaft für Mathematik und Datenverarbeitung (GMD)*, no. 448, June 1990.

[LNP87] von Luck K.; Nebel B.; Peltason Ch. and Schmiedel A.: The anatomy of the BACK-System. KIT-BACK report, Technische Universität Berlin, 1987.

[MAR88] Marcus, S.: SALT: a knowledge-acquisition tool for propose-and-revise systems. In *Automating Knowledge Acquisition for Experts Systems*, S. Marcus, Kluwer Academic Publisher, Boston, 1988.

[MCD87] Mc Dermott, D.: (reply(replies(critique(pure reason)))). In *Computer Intelligence 3*, 1987, 223-227.

[MKK86] Mitchell, T.M.; Keller, R.M.; and Kedar-Cabelli, S.T.: Explanation-based generalisation: a unifying view. In *Machine Learning*, no. 1, 1986, 47-80.

[MMS90] Mitchell, T.M.; Mahadevan, S.; and Steinberg, L.I.: LEAP: a learning apprentice system for VLSI design. In *Machine Learning. An Artificial Intelligence Approach, Vo. III*, Y. Kodratoff et.al. (eds.), Morgan Kaufmann Publisher, Los Altos CA, 1990.

[MOE87] Moerkotte,G.: Inkonsistenzen in deduktiven Datenbanken. Informatik-Fachberichte 248, W.Brauer (Hrsg.), Springer-Verlag, Berlin, 1987.

[MOR87] Morik, K.: Sloppy modeling. In *Knowledge Representation and Organization in Machine Learning*, K. Morik (ed.), Springer Verlag, Berlin, 1987.

[MOS83] Moser, M.G.: An overview of NIKL, the new implementation of KL-ONE, Research in Knowledge Representation for Natural Language Understanding. Bolt Beranek Newman Inc., Annual Report, no. 5421, 1983.

[MUS89] Musen, M.: *Automated Generation of Model-Based Knowledge-Acquisition Tools, Research Notes in Artificial Intelligence*, Pitman Publishing, London, 1989.

[NEW82] Newell, A.: The knowledge level, *Artificial Intelligence*, Vol. 18, 1982, 87-127.

[QUI90] Quinlan, J.R.: Probabilistic decision trees. In *Machine Learning. An Artificial Intelligence Approach, Vo. III*, Y. Kodratoff et.al. (eds.), Morgan Kaufmann Publisher, Los Altos CA, 1990.

[ROS77] Ross, D.T.: Structured Analysis (SA): a language for communicating ideas. IEEE Transactions on SE, vol. SE-3, no. 1, January 1977, 16-34.

[SBJ87] Soloway, E.; Bachant, J.; and Jensen, K.: Assesing the maintainability of XCON-in-RIME: coping with the problems of a VERY large rule-base. In *Proceedings of the 6th National Conference on Artificial Intelligence AAAI'87 (Seattle, Washington, July 13-17)*, 1987, 824-829.

[STM86] Stepp, R.E.; and Michalski, R.S.: Conceptual clustering: inventing goal-oriented classifications of structured objects. In *Machine Learning. An Artificial Intelligence Approach, Vo. II*, R.S. Michalski et.al. (eds.), Morgan Kaufmann Publisher, Los Altos CA, 1986.

[TEH77] Teichroew, D.; and Hershey, E.A.: PSL/PSA: a computer-aided technique for structured documentation and analysis of information processing systems. In *IEEE Transactions on software engineering*, vol. SE-3, no.1, January 1977.

[ULL89] Ullman, J.D.: *Database and Knowledge-Base Systems*, vol. 2, Computer Science Press, New York, 1989.

[WET90] Wetter, T.: First order logic foundation of the KADS conceptual model. In *Current Trends in Knowledge Acquisition*, B. Wielinga et. al. (eds.), IOS Press, Amsterdam, 1990.

[WIS90] Wielinga, B.; and Schreiber, G.: KADS: model based KBS development. In Proceedings of the 14th German Workshop on Artificial Intelligence GWAI-90 (Eringerfeld, September 1o-14), 1990, 322-333.

[WSG89] Wielinga, B.; Schreiber, G.; and de Greef, P.: Synthesis Report. Esprit Project P1098, UvA-Y3-PR-001, University of Amsterdam (The Netherlands), 1989.

[YOU89] Yourdan, E.: *Modern structured Analysis*, Prentice Hall, Englewood Cliffs, 1989.

APPENDIX - A FORMAL DEFINITION OF THE DOMAIN LAYER LANGUAGE PRIMITIVES

1. The Language Primitives

Def 1: (relational type definition)

i) A **relational type definition** is a tuple

$$R(t_1:R_1, ..., t_n:R_n)<a_1:\chi_1, ..., a_m:\chi_m>$$

R is called the **name** of the relational type. Each $t_i:R_i$ is called a **component** of the relational type, where t_i is the **name** of the component, and R_i is the **type name** of the component.

Each $a_j:\chi_j$ is called an **attribute** of the relational type, where a_j is the **name** of the attribute, and χ_j is the **type name** of the attribute.

ii) Let M be a set of relation type definitions, N_1 the set of all names of the relation types in M and N_2 the set of the type names of attributes which occur in relation type definitions of M.

The **dependency graph** DG_M is the set

$$DG_M = \{(R_i, R_j) \mid R_i \in N_1 \text{ and } R_j \text{ is a type name of a component of } R_i\}$$

A **schema definition** SD is a set of relation type definitions such that

1. Any type name R of a component of a relation type definition of SD must either be equal to "All" or R must appear as a name of a relation typewithin SD.
2. The names of the relation types are mutually different.
3. $All \notin N_1$
4. DG_{SD} is acyclic, i.e. DG_{SD} does not contain a sequence (R_1,R_2), (R_2,R_3), ..., (R_k,R_1).

Note

A schema definition is an "empty schema" for the occuring relationships on the domain layer. In part ii) of the definition above, condition 1) means that every type name must be defined in the schema definition except "All". Condition 2) means that there is exactly one definition for each type name. Condition 4) means that no type name is defined by itself.

Def 2: (domain, all possible instances, instance)

i) Let $O := \{(o_1),...,(o_k)\}$ be a finite set, the set of objects. Let SD be a schema definition, N_1,N_2 as above. Let $D = \{d_1, d_2, ...\}$ and $D' = \{d_1', d_2', ...\}$ be nonempty sets of finite sets, called the **domains**. Each d_i is a set of (nested) tuples with elements in O.
Let $\mathbf{dom_1}$ and $\mathbf{dom_2}$ be functions which map the names and respectively the attributes of SD to the corresponding sets of domains:
$dom_1: N_1 \cup \{All\} \rightarrow D$
$dom_2: N_2 \rightarrow D'$
and $dom_1(All) = O$.
These mappings are defined in such a way that for a relation type
$R(t_1:R_1, ..., t_n:R_n)<a_1:\chi_1, ..., a_m:\chi_m>$

$$dom_1(R) = \{(r_1, ...,r_n) \mid r_i \in dom_1(R_i) \text{ for all } i = 1,...,n\}$$

holds.

ii) For each $R \in N_1$ and the corresponding relation type $R(t_1:R_1, ..., t_n:R_n)<a_1:\chi_1, ..., a_m:\chi_m>$ we define a function f_R, which maps the elements of dom(R) to a tuple of attributes:

$$f_R: dom_1(R) \rightarrow dom_2(\chi_1) \times ... \times dom_2(\chi_m).$$

f_R is called the attribute function of R.
The set of **all possible instances** R^i_{all} of the relation type with the name R is defined by:
$R^i_{all} := \{(r_1, ...,r_n)<v_1, ...,v_m> \mid (r_1, ...,r_n) \in dom_1(R), f_R(r_1, ...,r_n) = (v_1, ...v_m)\}$
An **instance** of R must be an element of R^i_{all}. The set of all current instances of R is denoted by R^i.

Note:

Presently, D' contains only one set d' which is the set of all alphanumerical strings with finite length. Every set in D' includes "unknown".

Def 3: ("$\stackrel{\pm}{=}$","$\subseteq'$","$\sqsubset$")

Let I be the set of all instances of a schema definition : $I = \bigcup_{R \in N_1} R^i$

i) The function g is defined as follows:

$g: I \rightarrow D$

$g((t_1:i_1, ..., t_r:i_r)<a_1:v_1, ..., a_h:v_h>) = (t_1:i_1, ..., t_r:i_r)$.

We define a function f on D by

$f(e) = (e)$.

By means of these function we define the relation $\stackrel{\pm}{=}$ on I inductively as follows:

Let $i_1 = (t_1:i_{11}, ..., t_n:i_{1n})<a_{11}:v_{11}, ..., a_{1m}:v_{1m}>$ and

$i_2 = (t_1:i_{21}, ..., t_n:i_{2n})<a_{21}:v_{21}, ..., a_{2k}:v_{2k}>$.

$i_1 \stackrel{\pm}{=} i_2$ iff

1. $\exists\ p, q \in \{0, 1, 2, ...\}$ such that $f^p(g(i_1)) = f^q(g(i_2))$

or

2. $i_{1j} \stackrel{\pm}{=} i_{2j}$ for all $j = 1, ..., n$

ii) Additionally, we introduce the relation $\subseteq'$ on N_1:

Let $R_1, R_2 \in N_1$ with

$R_1(t_{11}:R_{11}, ..., t_{1n}:R_{1n})<a_{11}:\chi_{11}, ..., a_{1m}:\chi_{1m}>$

$R_2(t_{21}:R_{21}, ..., t_{2n}:R_{2n})<a_{21}:\chi_{21}, ..., a_{2k}:\chi_{2k}>$

$R_1 \subseteq' R_2$ iff $\forall\ i_1 \in R^i_1\ \exists\ i_2 \in R^i_2$ such that $i_1 \stackrel{\pm}{=} i_2$

iii) The relation $\sqsubset'$ on N_1 is defined inductively as follows:

1) If $R_1(t:R_2)<a_{11}:\chi_{11}, ..., a_{1m}:\chi_{1m}>$, then $R_1 \sqsubset' R_2$.

2) If $R_1 \sqsubset' R_2$ and $R_2 \sqsubset' R_3$ then $R_1 \sqsubset' R_3$

3) If $R_1(t_{11}:R_{11}, ..., t_{1n}:R_{1n})<a_{11}:\chi_{11}, ..., a_{1m}:\chi_{1m}>$ and
$R_2(t_{21}:R_{21}, ..., t_{2n}:R_{2n})<a_{21}:\chi_{21}, ..., a_{2k}:\chi_{2k}>$ and
$\exists\ i \in \{1, ..., n\}$ with $R_{1i} \sqsubset' R_{2i}$ and for all $j = 1, ..., n$: $R_{1j} \sqsubset' R_{2j}$ or $R_{1j} = R_{2j}$
then $R_1 \sqsubset' R_2$

The relation $\sqsubset$ is defined as follows:

$R_1 \sqsubset R_2$ iff $R_1 \sqsubset' R_2$ and $R_1 \subseteq' R_2$.

Corollary: $\stackrel{\pm}{=}$ is an equivalence relation on I.

Note:

The function "g" in part i) of the above definition "forgets" the attributes of an instance, "f" encloses its argument in brackets. "$\subseteq'$" is a partial, reflexive ordering on N_1. It means that the instances of one relation type are a subset of another set of instances. Note that only brackets are added which include a single component or the whole instance. "$\sqsubset$" defines a partial, irreflexive ordering on N_1. It is clear from the definition that if $R_1 \sqsubset R_2$ then $R_1 \subseteq' R_2$, but

you cannot conclude the other way round: The instances of a relation type can be a subset of the instances of another relation type "by chance".

All relation types can be an element of an **is-a hierarchies "<"**. The "is-a" hierarchy introduces sub and super relation types in combination with attribute inheritance among them. Therefore for a subrelation type only the additional attributes must be defined in its definition. We write

$R_1 < R_2$

if the relation type "R_1" is a subrelation type of the relation type "R_2". In this case "R_1" inherits all attributes of "R_2" and every instance of "R_1" inherits all attributes values of the equivalent ($\hat{=}$) element of "R_2". Two necessary conditions for such a is-a link between two relation types exists:

1. $\neg R_2 \sqsubset R_1$
2. $R_1 \subseteq' R_2$

2. The Modelling Primitives

a. Concepts

A **concept** concept_name is represented by a relation type definition which has the following form:

concept_name(t:R)<a_1:χ_1, ..., a_m:χ_m>

and R is a name of a concept. "All" is the name of a concept.

b. Relationships

A n-ary **relationship** relation_name between n concepts C_1, ..., C_n is represented by a relation type definition which has the following form:

relation_name(t_1:C_1, ..., t_n:C_n)<a_1:χ_1, ..., a_m:χ_m>,

where C_1, ..., C_n are names of concepts.

A **relationship** relation_name´ between arbitrary relations R_1, ..., R_n is represented by a relation type definition which has the follwowing form:

relation_name´ (t_1:R_1, ..., t_n:R_n)<a_1:χ_1, ..., a_m:χ_m>

A **is-a relationsship** between two relationships rel_name_1 and rel_name_2 is represented by an "is-a" hierarchy which has the following form:

$rel_name_1 < rel_name_2$

c. Instances: An **instance** of a relationship which is represented by the relation type R is an element of R^i.

d. Intensional descriptions of a relationship or a concept: is possible with KARL clauses, e.g.

$R_1(t_m{:}X) \leftarrow R_2(t'_1{:}X) \,\&\, R_3(t'_3{:}X).$
$R_1(t_m{:}X)\langle a{:}Y\rangle \leftarrow R_2(X)\langle b{:}Y\rangle.$
$R_1(t_m{:}(X_1, ..., X_k)) \leftarrow R_2(t'_1{:}X_1, ..., t'_k{:}X_k).$

e. Extensional descriptions of a relationship or a concept: is an instance, e.g.:

$R(t_1:(r_{11}, ..., r_{1w}), ..., t_m: (r_{m1}, ...r_{mw}), t_{m+1}:r_i)<a_1:13,a_n:50>$.

f. Constraints:

In addition to the introduced primitives, constraints may be expressed with KARL clauses. Functions are not allowed in constraints. Variables in constraints may be universally or extensionally quantified. If the variables are not universally quantified the quantification has to be described with quantifiers. Functions are not allowed in constraints. Constraints have to be "range restricted" which means that for every universally quantified variable there exists at least one negative literal in each disjunction of the conjuctive normal form of the constraint containing the variable. For every existentially quantified variable occuring in a negative literal in the conjunctive normal form of the constraint there exists a disjunction containing only positive literals each containing the variable.

KASE Project toward Effective Diagnosis System Developments

Dai Araki Shoichi Kojima
Systems & Software Engineering Laboratory
TOSHIBA Corporation
70, Yanagi-cho, Saiwai-ku, Kawasaki, 210 JAPAN
(e-mail: araki%ssel.toshiba.co.jp@uunet.uu.net)

Abstract

The Knowledge Acquisition Support Environment (KASE) and a case study on expert system development are described. DiPROS is a diagnostic expert system building tool, which has a knowledge base editor and an inference engine specialized for diagnostic tasks. DiKAST is an interactive knowledge acquisition (KA) tool, which guides repairs on defects in the knowledge base. In order to build this KA facility, many kinds of KA interview strategies were formalized. We applied KASE tools in building a defect diagnosis system for a color picture tube manufacturing plant. The prototype system was able to be developed in a short period of time by the domain experts alone, who had no previous experience in computer programming.

1 Introduction

1.1 KASE Framework

This paper shows development in the project for Knowledge Acquisition Support Environment (KASE) for diagnostic problems. The goal for this project was to reduce the bottleneck in knowledge acquisition (KA), and to construct an expert system development environment, where special knowledge engineers are not needed.

The development of the basic framework for KASE was started from the construction of a diagnosis problem-solving model. The importance of the appropriate problem-solving model was discussed in (Musen,1989; Karbach, Linster and Voss, 1990; Gruber and Cohen, 1987; Bylander and Chandrasekaran, 1988). It is known that the KA task can be made more effective by limiting the target problem area and by building a conceptual model of the knowledge base, called the *expertise model*.

The problem-solving knowledge can be segregated into two categories and elicited separately from the domain experts. First is the basic framework of the problem-solving activities, called the *problem-solving method*, which means a rough sketch of the problem-solving flow chart. This knowledge category includes those kinds of procedures which are used in

problem-solving and how they are connected as a whole. The other category is the various *fragmentary knowledge*, which is used in each problem-solving procedure. In a limited problem area such as diagnosis, the problem-solving method can be considered as application domain independent knowledge, which can be applied regardless of the diagnosis target. Therefore, we made an *expertise model* of the diagnostic task. In addition to the problem-solving method, "What kinds of fragmentary knowledge are needed?", "How can each kind of fragmentary knowledge be represented?" and "How can each kind of fragmentary knowledge be used during the problem-solving?" was also included in the expertise model, as application domain independent knowledge.

By establishing an appropriate expertise model for the target problem a priori, the KA task for a new diagnosis system development can be accomplished by obtaining fragmentary knowledge contents from domain experts. There are two merits in constructing an appropriate expertise model as KASE framework.

- An expertise model is useful in describing problem-solving behavior at a more adequate and abstract level than that offered by a general purpose knowledge representation language.
- Models guide the KA process and structure the knowledge base.

The developments of many KA tools, such as MOLE, OPAL or SALT (Eshellman, 1988; Musen, Fagan, Combs and Edward, 1987; Marcus, 1988) take a model-driven approach, and each of the KA tools above have formalized interactive KA manners. However, there are several problems in applying these existing KA tools to practical domains and establishing an expert system development environment, where special knowledge engineers are not needed. In the next subsection, we discuss several requirements for diagnostic KASE and present an overview of approaches used which meet these requirements.

We developed two KASE tools on the expertise model constructed (→ Fig.1). One is a diagnostic task-specific problem solver, named DiPROS (Diagnostic Problem Solver). The other is an interactive KA tool, named DiKAST (Diagnostic Knowledge Acquisition Support Tool) which contains many kinds of KA interview strategies.



FIG. 1. KASE framework.

1.2 Requirements and Overview for Our Approaches

In constructing a diagnosis expertise model and KASE tools, the following requirements were considered.

The model should be specific in a diagnostic problem area, and independent from application domains.

By limiting the target application domain, it become possible to define some knowledge representation styles, which can be easier for domain experts to understand. For example, OPAL (Musen, Fagan, Combs and Edward, 1987) is a cancer treatment domain specific KA tool, and ROGET (Bennet, 1985) is specialized for medical diagnosis. However, our targets include many application domains, such as a diagnosis on color picture tube, HDD or traffic signal equipment, so the expertise model should be more generally applicable than the above domain specific KA tools. In our KASE tools, all of the diagnostic knowledge is represented in the form of a network and several kinds of tables, and vocabulary specific to that diagnosis is used. In this way, the KASE tools provide a more adequate and abstract level representation style for describing diagnostic knowledge than the implementation level knowledge representatives such as production rules or frames. Thus, diagnosis domain experts, who are not skilled programmers, are able to describe their knowledge.

Strategic knowledge should be represented easily.

Diagnosis is often treated as purely a classification problem. However, in many application domains, strategic knowledge underlies the know-how in effective diagnosis when choosing the next question to ask or which diagnostic test to run (Gruber, 1989). This strategic knowledge controls time, cost and the effort necessary for the diagnosis. However, the diagnosis expertise models of MOLE (Eshellman, 1988) (cover-and-differentiate) or CSRL (Bylander and Mittal, 1986) (heuristic classification) and many other classification KA tools, such as ETS (Boose, 1985), AQUINUS (Boose and Bradshaw, 1987), and KRITON (Diederich and Linster, 1989) do not consider strategic knowledge.

DiPROS can represent two kinds of strategic knowledge. Furthermore, in order to acquire such strategic knowledge automatically, we take a different approach from ASK (Gruber, 1989) or ODYSSEUS (Wilkens, Clancey and Buchanan, 1987). DiPROS can search possible diagnosis answers sequentially and exhaustively. This strategic knowledge affects answering order. If the user desires to change this order, such as "The second answer is more preferable than the first one", then DiKAST analyzes the previous inference path and proposes some plans to modify the strategic knowledge.

Diagnostic Knowledge should be systematized and reused easily in order to accommodate multiple knowledge sources.

The most important diagnosis knowledge is the relationships among various events, which happen to the diagnosis target. In existing diagnostic problem solvers, such as MOLE and CSRL, this knowledge is often represented as a single relational network for the diagnostic events. While this representation style is suitable for interpreting in the inference engine, it doesn't meet the above requirement. We propose a new knowledge representation style, called the *multi-relational network model*, and draw these relationships in a three-dimensional diagram on a screen. This style is an extension of the symptom-component approach (Bradley and Briggs, 1989).

Automatic interview facilities should be provided to complete and maintain the knowledge base.

Similar to ETS, AQUINUS, MOLE and so on, DiKAST is an interactive KA tool. Experienced knowledge engineers use many heuristic interview strategies. The interview techniques for DiKAST are based on formalization of these heuristic interview strategies. The KASE tools use a expertise model with a high expression ability, so that the KA ability for DiKAST is enhanced. The explanatory comments or conversation statements, which DiKAST produces, are easier for domain experts to understand.

Good man-machine interfaces should be provided.

The KASE tools are built on the Open Windows System with OPEN LOOK interface specifications. A good look and feel in the interface helps domain experts input their knowledge more easily. Moreover, the KASE tools have many useful facilities for completing the diagnosis system in real-world use. For example, the diagnostic data can be retrieved through files or user defined C language functions.

2 Expertise Model

2.1 Problem-Solving Method and Fragmentary Knowledge

In our model, the diagnostic problem is considered as a classification problem. In other words, all possible diagnosis results are enumerated in the knowledge base, and the goal of the problem-solving is to explain any unusual symptom by classifying the unusual symptom and select an appropriate explanation from the enumerated diagnosis results.

As shown in Fig.2, the problem-solving method for the diagnosis model is represented by a loop of three basic procedures. First, an initial unusual symptom, such as "The robot can't grasp the target object", is input into the Hypothesis Making step, where it is treated as the first target event for explanation. The procedures are as follows.

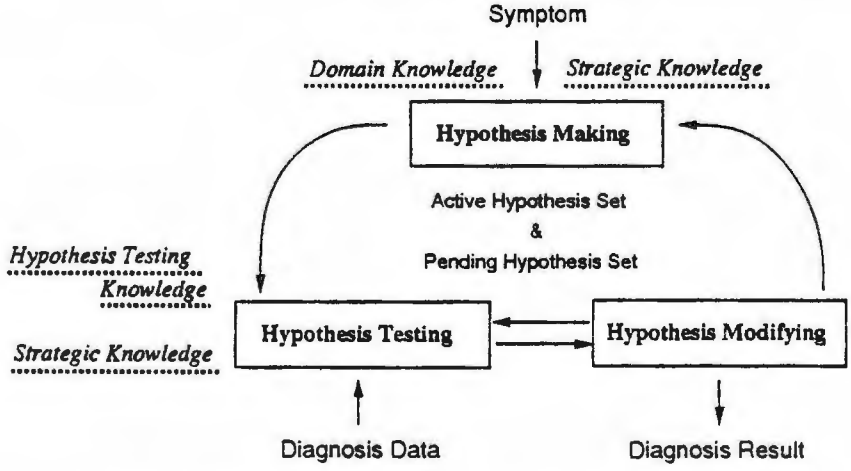


FIG. 2. An expertise model of the diagnostic task.

Hypothesis Making generates some hypotheses in order to explain the target event. There are two hypothesis sets, an active hypothesis set and a pending hypothesis set. The active hypothesis set includes the hypotheses currently treated as the target for testing in the Hypothesis Testing procedure. The pending hypothesis set includes the hypotheses currently excepted from the target for testing. Each hypothesis is distributed by evaluating relational weights. The modification of a hypothesis set occurs in Hypothesis Modifying. This mechanism realizes diagnostic strategies such as "The hypotheses with low frequencies should be postponed for later consideration".

Hypothesis Testing collects some diagnosis data and differentiates between individual active hypothesis. For the active hypothesis set, the differentiating method with the least cost for judging the correctness or incorrectness of an active hypothesis is tried. These methods are represented by the conditions for the diagnosis data. For the selected method, all of the diagnosis data specified in the condition are checked, to see whether or not they have been gathered. If any diagnostic data is missing, it is then asked for outside. If all the possible differentiating methods are executed, but the hypothesis can't be differentiated, backtracking occurs.

Hypothesis Modifying modifies the contents of two hypothesis sets according to the following algorithm:

1. Negated hypotheses are deleted.
2. If there is a confirmed hypothesis:
   (a) If it is one of the final explanations for diagnosis, inference is stopped, and it is output as the diagnosis result.
   (b) Otherwise, the target event for explanation is exchanged, two hypothesis sets are cleared, and Hypothesis Making is implemented again.
3. If there are no active hypotheses, some pending hypotheses are moved to the active hypothesis set.
4. If there are no hypotheses in either hypothesis sets, backtracking occurs.
5. Hypothesis Testing is implemented again.

In this model, an explanation for unusual symptoms is searched for in the establish-and-refine fashion. Three basic procedures are circularly carried out and the target event for explanation is exchanged successively. A backtracking mechanism enables the process to backtrack to some previous inference world and to retry in testing other hypotheses.

As shown in Fig.2, we classified fragmentary knowledge into the following three classes, and defined the knowledge representation styles for each class.

Domain Knowledge decides which hypothesis should be made to explain the target event. We propose a multi-relational network model to represent domain knowledge. Details on the multi-relational network will be mentioned in Section 2.2.

Hypothesis Testing Knowledge means how to judge the correctness or incorrectness of a target hypothesis. The method for judging whether a hypothesis is true or not is called

a *diagnostic test.* Diagnostic tests specify what data are needed to confirm or negate a hypothesis. Each diagnostic test is represented by a condition statement and an evaluation method. The condition statement includes several diagnosis data values or ranges which must be satisfied, such as "The value of the release control signal from the arm processor is OFF". The evaluation method prescribes how to treat the target hypothesis (to be confirmed or negated) when the condition is proven to be satisfied or unsatisfied.

Strategic Knowledge controls the sequence of diagnostic information retrieval in Hypothesis Making and Hypothesis Testing. Two kinds of strategic knowledge are defined in this model: "Which hypothesis should be considered by preference?" and "Which diagnostic test should be applied by preference?". The former is used to distribute hypotheses in two hypothesis sets. The latter is used to select one diagnostic test from among the candidates. Strategic knowledge is represented by the numerical values.

2.2 Multi-relational Network

We applied this expertise model to represent the actual behavior of domain experts in the course of developing several diagnostic systems, and analyzed the explanations produced in each procedure cycle. As a result, we found that there are two kinds of explanations usually produced from the basic procedure loop.

One is the kind of explanation where "Event A is caused by Event B". This is called causal inference. Causal inference is the most typical inference in diagnostic problem-solving. The other is the kind of explanation where "Event A can be made concrete or detailed by Event B". This kind of explanation also plays an important role in diagnostic problem-solving. "The engine fault proved to be a faulty spark plug." is one example. Almost all explanations are classified into these two categories. Therefore, the diagnostic hypotheses, which domain experts set forth to explain a target event, are derived from the following two expertise categories.

- Causal relationship
- Constitutional or conceptual classification relationship

Both kinds of knowledge are concerned with the relationships among various events, which happen to the diagnosis target. This relational knowledge is referred to as *domain knowledge.* Domain knowledge represents the original nature of the targets, but it should be modified or abstracted to suit the diagnosis.

We represented domain knowledge as a three-dimensional diagram, called a *multi-relational network.* In Fig.3, partial domain knowledge for the fault diagnosis for a robot arm is shown as a multi-relational network. Diagnostic events are grouped according to constitutional or conceptual classification hierarchies. Each event group is enclosed in a parallelogram and constitutes a plane. Individual relationships are represented by solid line links (Classification_Link). For example, the plane `Abnormal-Finger-Action` is the conceptual classification hierarchy for abnormal finger actions. `Abnormal-Finger-Action` can be classified into `Grasp-Impossible` and `Release-Impossible`. We defined *classified-lower* or *classified-upper* to represent relationships between events in the classification hierarchy. Causal relationships among the diagnostic events are represented by dotted line links (Causal_Link).

We defined *causal-lower* or *causal-upper* to represent causal relationships. For example, `Abnormal-Arm-Mechanism` has two causal-lower events: `Fg-Signal-Transfer-System` and `Fg-Driver-Circuit`.

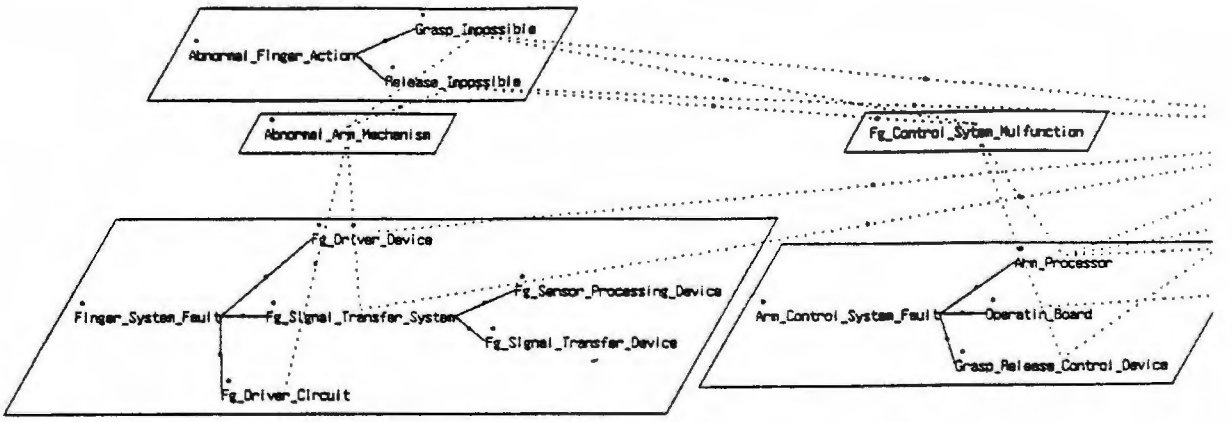


FIG. 3. A domain knowledge example.

There are two merits in making a distinction line between the two kinds of relationships for KA tasks.

- A multi-relational network is good for expressing and understanding domain knowledge, because there may be different knowledge sources for acquiring individual relational knowledge. For example, constitutional classification relationships among diagnostic events can be acquired from a circuit diagram or from specifications regarding the target.
- We were able to establish several interview strategies, which make the best of a multi-relational network. They have been implemented in DiKAST. These interview strategies can support abstracting or modifying initial domain knowledge and acquiring hypothesis testing knowledge suitable for diagnosis.

3 Diagnostic Problem Solver

DiPROS has two basic facilities for making diagnosis expert system development easier: a knowledge editor and an inference engine.

3.1 Knowledge Editor

DiPROS possesses a multi-window knowledge editor. One knowledge base editing scene is shown in Fig.4.

The **Network Definition Window** is used to describe domain knowledge in the form of a multi-relational network. The network editing operations can be accomplished by the pop-up menus. Main controls of DiPROS, such as "KB management" or "Execute Diagnosis", can also be manipulated from this window.

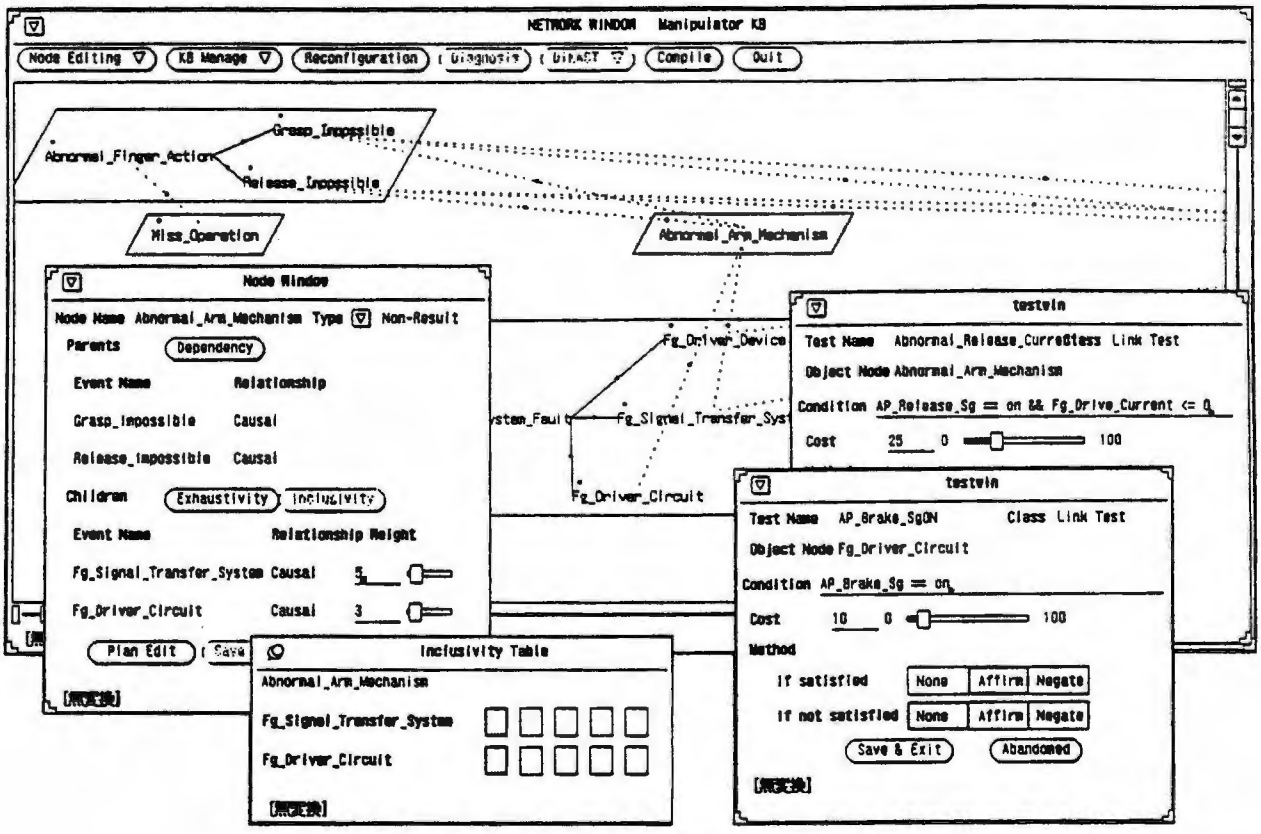


FIG. 4. One knowledge base editing scene.

The **Node Definition Window** is used to define some detailed attributes of the diagnostic event. The left-hand window in Fig.5 shows the node definition of `Abnormal-Arm-Mechanism`. This window displays its event type (Non-Result), list of associated parent events and child events. The user can select one event type from Symptom, Non-Result and Result. Three types of logical relationships can be defined between related events: *mutual dependency* between parent events, and *exclusivity* and *exhaustivity* between child events. Each logical relationship can be defined on the matrix window which can be opened by clicking a panel button. If it is defined that "`Grasp-Impossible` and `Release-Impossible` are in mutual dependency", then `Abnormal-Arm-Mechanism` would not be considered until both `Grasp-Impossible` and `Release-Impossible` are confirmed. If it is defined that "`Fg-Signal-Transfer-System` and `Fg-Driver-Circuit` are in exclusivity", then hypothesis `Fg-Signal-Transfer-System` would be negated when hypothesis `Fg-Driver-Circuit` is confirmed. If it is defined that "`FgSignal-Transfer-System` and `Fg-Driver-Circuit` are in exhaustivity", then hypothesis `FgSignal-Transfer-System` would be confirmed when hypothesis `Fg-Driver-Circuit` is negated.

The *relational weight* between child events determines the priority of hypothesis checking, that is, "Which hypothesis should be considered by preference?". This value is one kind of strategic knowledge. In Fig.5, the relational weights between children nodes are defined as 5 and 3 respectively, so that `Fg-Signal-Transfer-System` is treated as the more preferred explanatory hypotheses for `Abnormal-Arm-Mechanism` than `Fg-Driver-Circuit`.

The **Test Definition Window** is used to define one diagnostic test. Diagnostic tests are

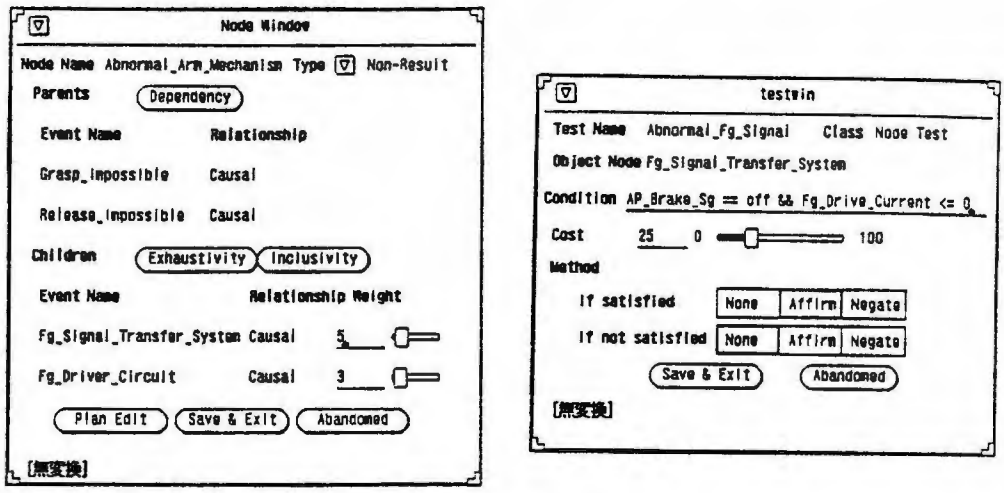


FIG. 5. Node Definition Window and Test Definition Window.

represented as ○ or ● on the network definition window. ○ means that no diagnostic tests are defined at this point, while ● means that some diagnostic tests are defined. There are two kinds of diagnostic tests: *node_test* and *link_test*. A node_test is a diagnostic test which can be applied without any prior condition, while a link_test has a prior condition. For example, in Fig.6, both NT_1 and NT_2 are node_tests (represented by ● on node H) which can be applied to test a hypothesis H at any time, while LT_1 is a link_test (represented by ● on link $E_1 \rightarrow H$) which can be applied only if the current target event for explanation is E_1. The right-hand window in Fig.5 shows the diagnostic test definition of `Abnormal-Fg-Signal`. The *execution cost* refers to the priority of this diagnostic test, and defines "Which diagnostic test should be applied by preference?".

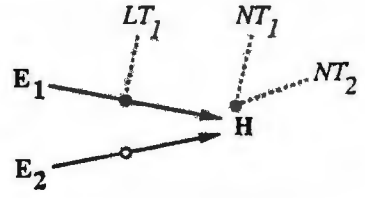


FIG. 6. Two kinds of diagnostic tests.

3.2 Inference Engine

DiPROS has a diagnostic inference engine, so it can be immediately applied as the diagnosis system. Fig.7 shows one diagnostic problem-solving scene. Diagnosis is started by requesting the first unusual symptom to be explained. Diagnosis is continued by inquiring about any necessary diagnosis data. These operations are basically accomplished through some kind of pop-up window in an interactive manner, but data retrieval from files or some user-defined C language functions are also available. The diagnostic trace is explained in the lower right-hand window in the figure, and is also displayed graphically in the network definition

window. In the network definition window, the diagnostic trace and current hypothesis are highlighted as the inference proceeds.

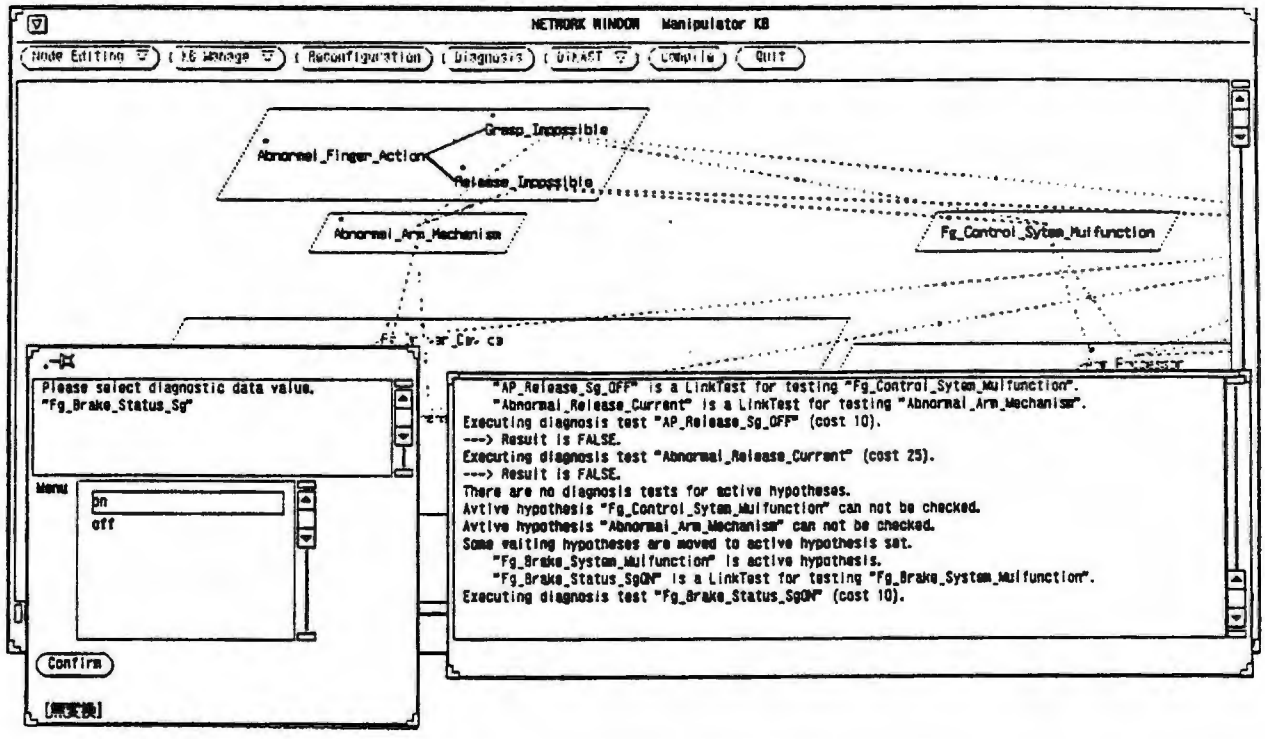


FIG. 7. One problem-solving scene.

4 Interactive Knowledge Acquisition Tool

DiKAST can be executed at any time from the KB editing screen of DiPROS. DiKAST seeks defects in the knowledge base, and starts KA interviews to correct them. There are three KA interview sessions: *Domain Knowledge Analysis*, *Hypothesis Testing Knowledge Analysis* and *Inference Path Analysis*.

4.1 Domain Knowledge Analysis

In the domain knowledge analysis, two kinds of defects, insufficiency and redundancy, are analyzed. DiKAST seeks defects in the domain knowledge and highlights event nodes in question on the network editor window (→ Fig.8). By selecting one of the highlighted event nodes, the KA interview session for the selected event node is started.

If some Non-Result node, named X, has no lower-relational event nodes, it would be unable to create any hypothesis to explain event X. This is an example of insufficiency in the domain knowledge. In this case, DiKAST determines that X may be a Result node and asks whether event X can be seen as one of the final explanations for diagnosis or not. If event X is confirmed not to be one of the final explanations, then DiKAST would inquire

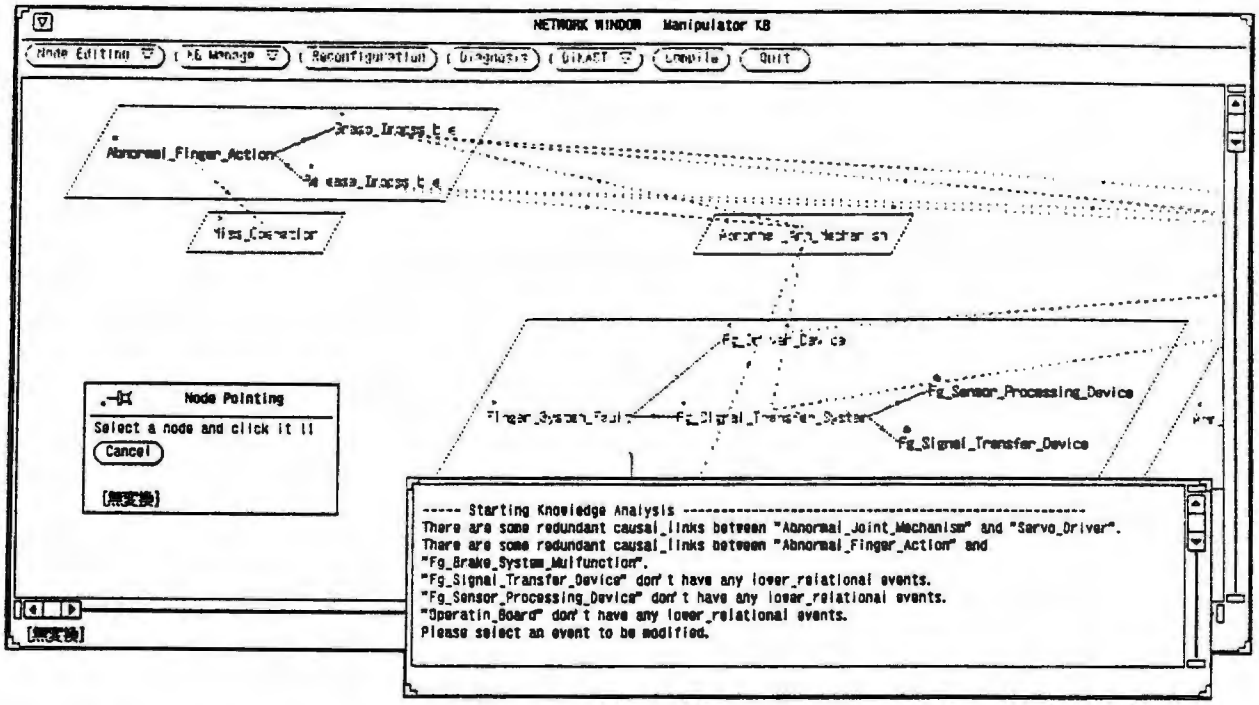


FIG. 8. Domain knowledge analysis. *Event nodes in question are highlighted, while others are displayed gauzily.*

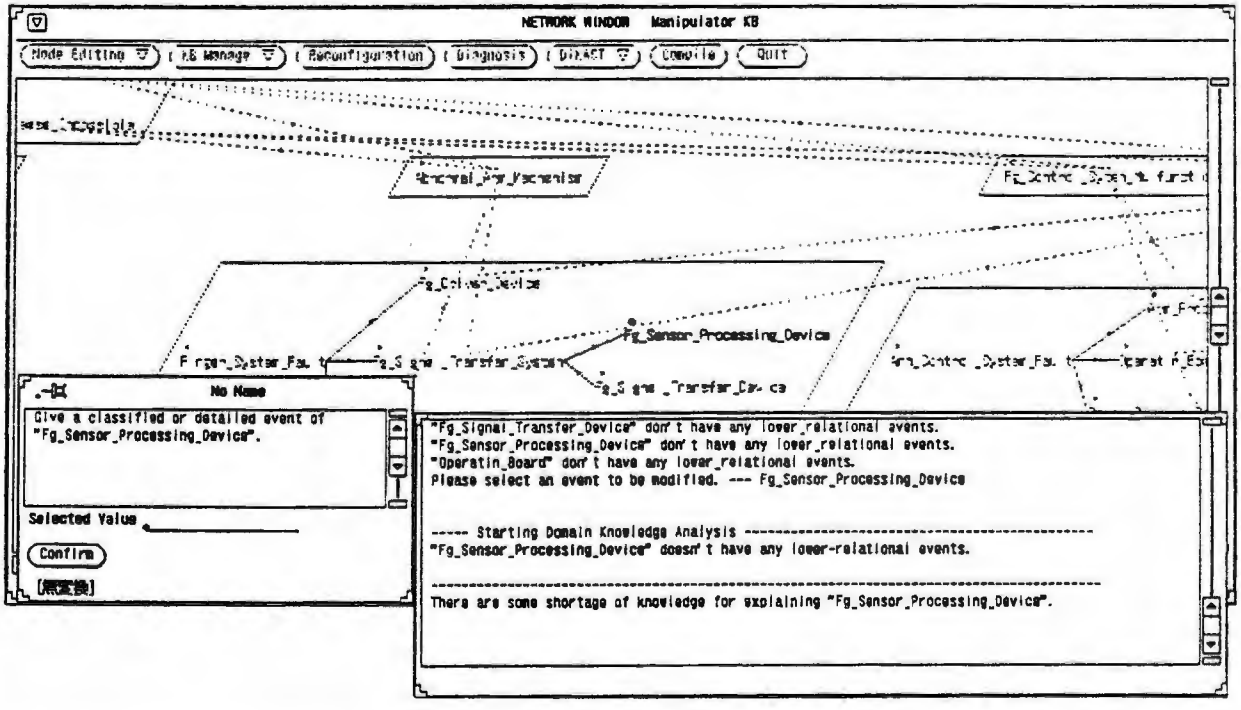


FIG. 9. Interview for insufficient domain knowledge. *DiKAST is asking about some classified-lower events for* Fg-Sensor-Processing-Device.

about some classified-lower events or causal-lower events in relationship to event X. Fig.9 shows the KA interview for the insufficient event `Fg-Sensor-Processing-Device`.

An example of redundancy in the domain knowledge is shown in Fig.10(A). Events X_1, X_2 and X_3 are classified-lower events of event X, but share the same causal-lower event Y. In this case, even if event X could be explained by some classified-lower events X_1, X_2 or X_3, there is no increase in information to justify fault hypotheses Y. In order to remove such redundancies, DiKAST proposes to change the relational links, as in Fig.10(B). Fig.11 shows the KA interview for redundancy in the domain knowledge.

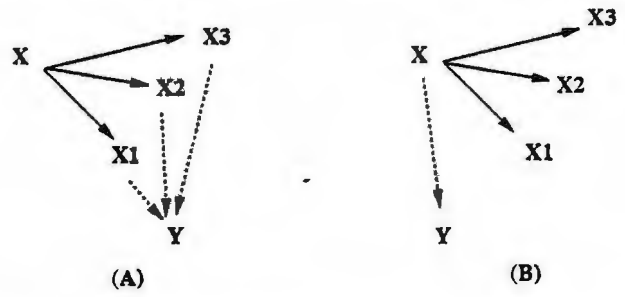


FIG. 10. Redundancy in the domain knowledge.

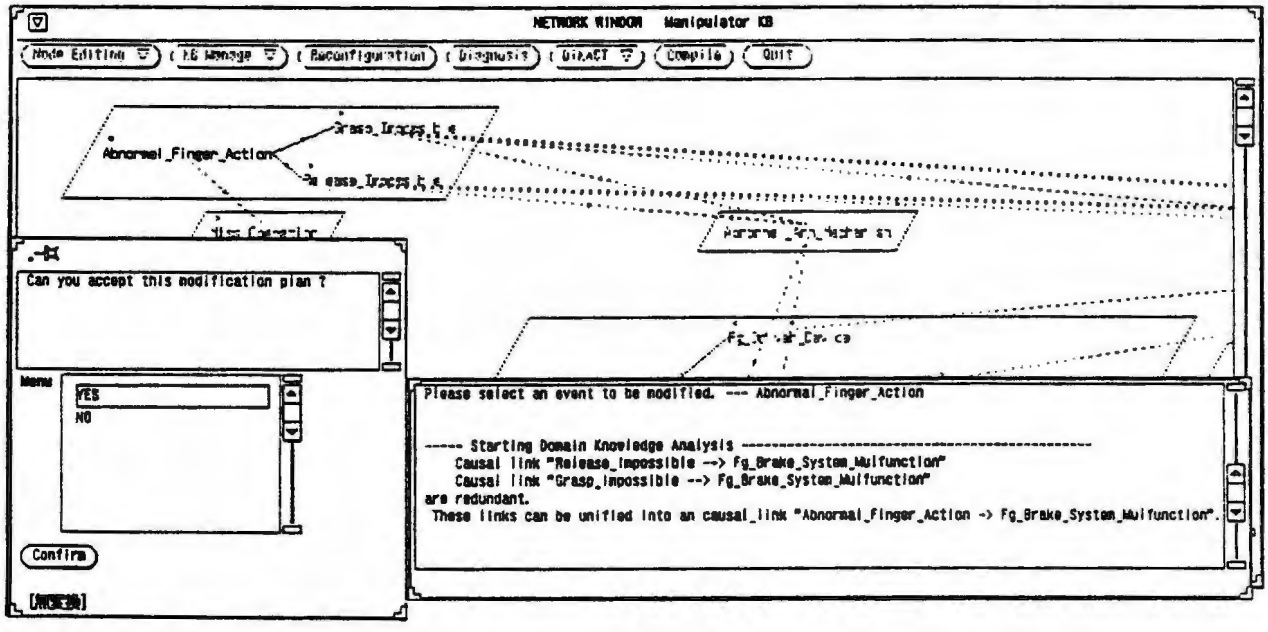


FIG. 11. Interview for redundant domain knowledge. *DiKAST points out that the relational links* `Grasp-Impossible` → `Finger-Brake-Fault` *and* `Release-Impossible` → `Finger-Brake-Fault` *are redundant, and proposes to unify them into* `Abnormal-Finger-Action` → `Finger-Brake-System-Malfunction`.

4.2 Hypothesis Testing Knowledge Analysis

The insufficiency of hypothesis testing knowledge is defined formally as in Fig.12. If there are no link_tests for the relational link $E_1 \rightarrow E_2$ and no node_tests for node E_2, it cannot be determined whether hypothesis E_2 is true or not for explaining E_1. DiKAST has several kinds of KA interview strategies for inquiring about missing hypothesis testing knowledge.

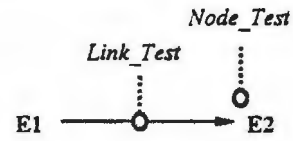


FIG. 12. Insufficiency of hypothesis testing knowledge.

Analogical Reuse

At first, DiKAST tries to acquire missing hypothesis testing knowledge through the *Analogical Reuse* strategy. If there are some structural similarities in the diagnostic tests, DiKAST proposes to reuse the same diagnostic tests as the method for justifying another hypothesis. In Fig.13(A), relational link $A \rightarrow n_1$ has a diagnostic test LT_a, and n_2 has a diagnostic test NT_a, but B has no diagnostic test for hypothesis n_1. In this case, it is proposed to use diagnostic test LT_a as the missing diagnostic test. If the user accepts this proposal, DiKAST modifies the domain knowledge, as in Fig.13(B).

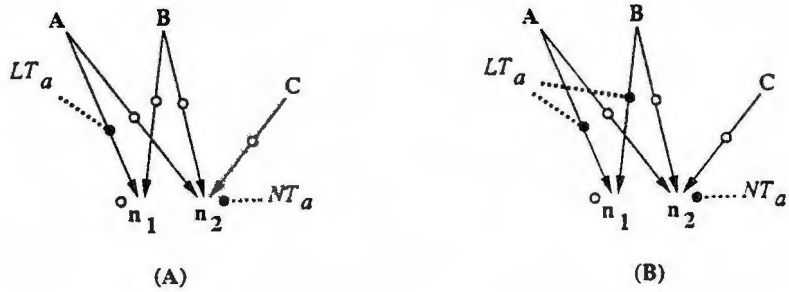


FIG. 13. Analogical reuse.

Direct Inquiry

If the analogical reuse fails, DiKAST would try the *Direct Inquiry* strategy. Direct Inquiry is the simplest interview method to acquire missing knowledge. In the situation in Fig.12, DiKAST inquires about missing hypothesis testing knowledge with the following questions: "Do you know how to justify E_2?" or "Do you know how to justify E_2 as the explanation for E_1?". Fig.14 shows one scene in the direct inquiry KA interview.

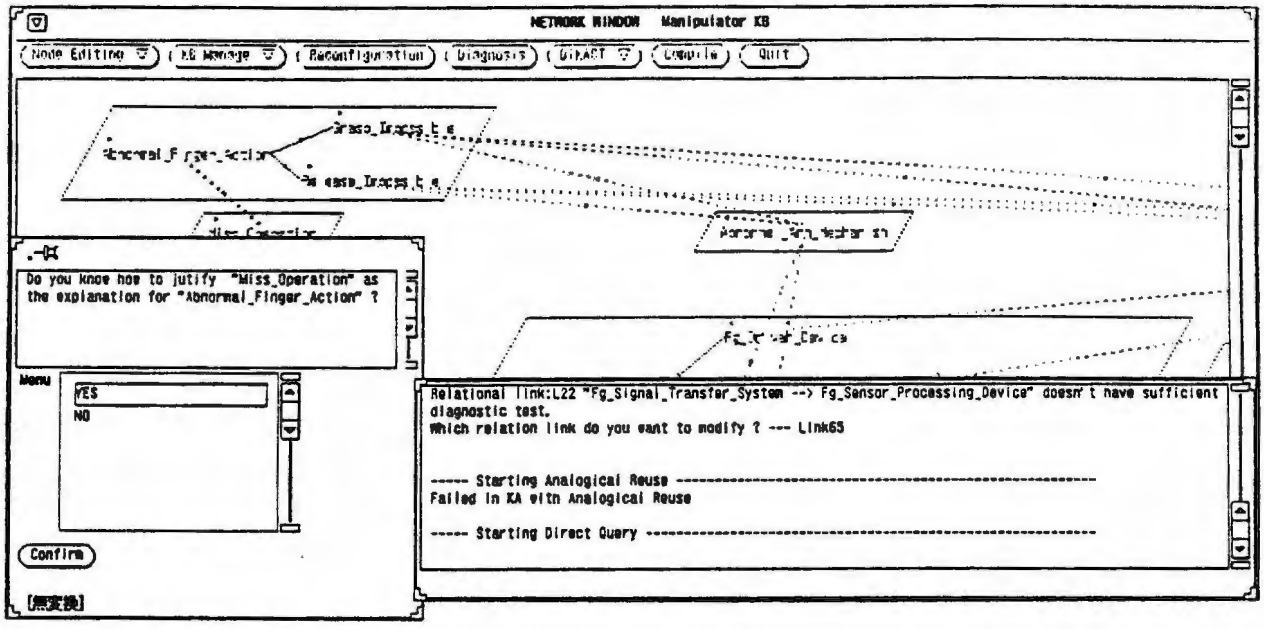


FIG. 14. Direct inquiry interview. *DiKAST is trying direct inquiry to find the missing diagnostic test for the relational link* `Abnormal-Finger-Action` → `Miss-Operation`.

Viewpoint Switching

If the direct query fails, DiPROS would take the *Viewpoint Switching* strategy. Consider a situation as in Fig.15(A), where DiKAST could not find some diagnostic tests for the relational link $X \rightarrow Y$ (link_test LT_y or node_test NT_y). DiKAST changes the viewpoint of the query and replaces X or Y with one of the more concrete or detailed events. For example, DiKAST replaces the hypothesis event from Y to Y_1 and changes the previous inquiry for NT_{y1} to "Do you know how to justify Y_1?" or for LT_{y1} to "Do you know how to justify the hypothesis Y_1 as the explanation of X?".

A similar inquiry alternation is implemented by replacing the hypothesis event from Y to Y_2 and inquiring about NT_{y2} or LT_{y2} (→ Fig.15(C)). After these interviews, DiPROS adds a new diagnostic test and modifies the domain knowledge. Fig.15(D) shows an example of the modified domain knowledge. In Fig.15(D), $X \rightarrow Y$ is changed to $X \rightarrow Y_1$, because only NT_{y1} could be acquired through the viewpoint switching interviews.

Similar viewpoint switching interviews and domain knowledge modifications can be achieved by replacing the explanatory event X with a more concrete event X_1 or X_2.

4.3 Inference Path Analysis

In the case of some diagnosis error, for example, if an undesirable diagnosis answer is obtained, the last type of KA interview session, called *Inference Path Analysis*, would be used to correct the knowledge base. DiKAST analyzes the wrong inference trace for the undesirable answer and proposes some plans for repairing the knowledge base defects.

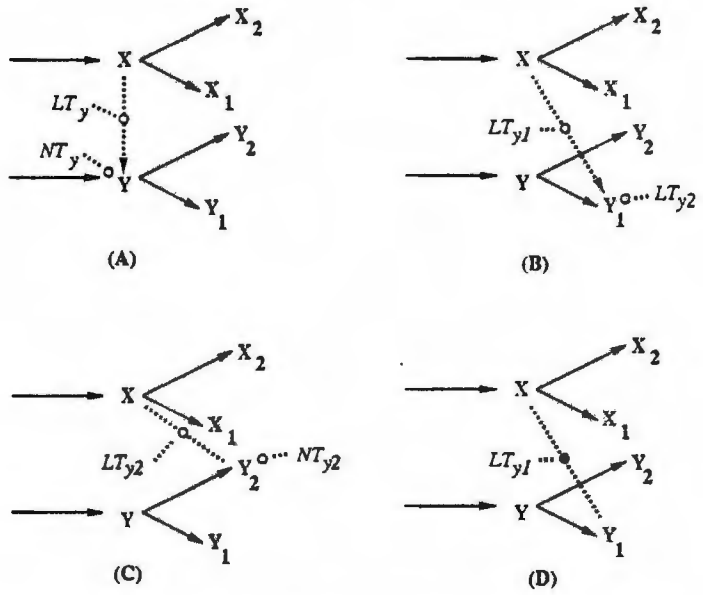


FIG. 15. Viewpoint switching.

Fig.16 shows one scene of the Inference Path Analysis interview. The KA interview would be comprised of the following steps:

1. A correct diagnostic answer node, that the user wishes to obtain, is asked for.
2. The domain knowledge is analyzed from the node representing the desired node in the opposite direction from the route in which the network was traced in the previous diagnosis, and detection of a diverging point (DP) at which both routes split (→ Fig.17) is tried.
3. The decision, justifying why such a direction was selected at the detected diverging point, is shown.
4. Some possible knowledge modification plans to correct the diagnostic trace as the direction toward the desired solution node are presented.
5. It is asked whether the decision justification is correct or not.
6. If it is found that the justification is incorrect, the user is required to choose one of the presented plans. Otherwise, go back to step 2.
7. The knowledge base is modified in accordance with the selected plan, and the KA interview is terminated.

In this KA interview, many kinds of incorrectness in the knowledge base are synthetically analyzed. The strategic knowledge in the knowledge base is also included in the modification target. The KA interview for the strategic knowledge proceeds as follows. If the user desires to change the order of answering, such as "The second answer is preferable to the first one", DiKAST proposes some modification plans, such as "Change the cost of diagnostic test A

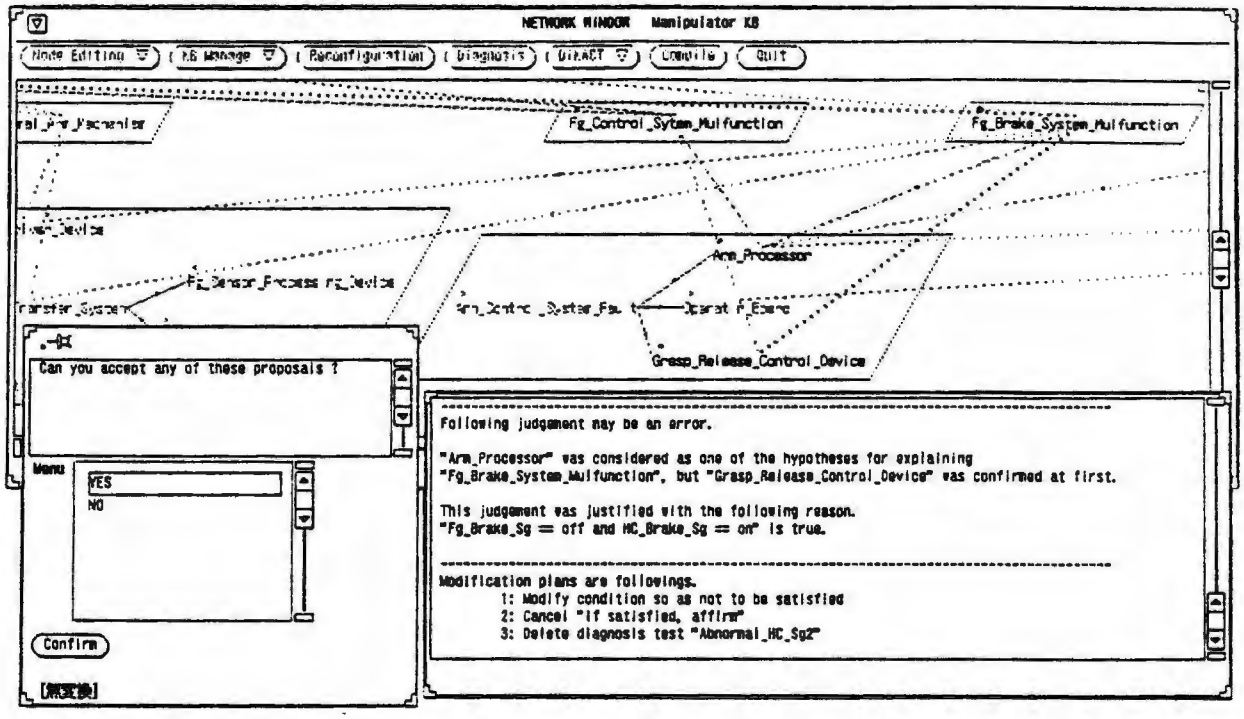


FIG. 16. Inference path analysis interview. *DiPROS inferred that "*`Release-Impossible` *is caused by* `Finger-Control-Parts`*", but the user indicates that "The correct answer is* `Arm-Processor`*". DiKAST found one possibility of an inference error and proposed three kinds of modification plans.*

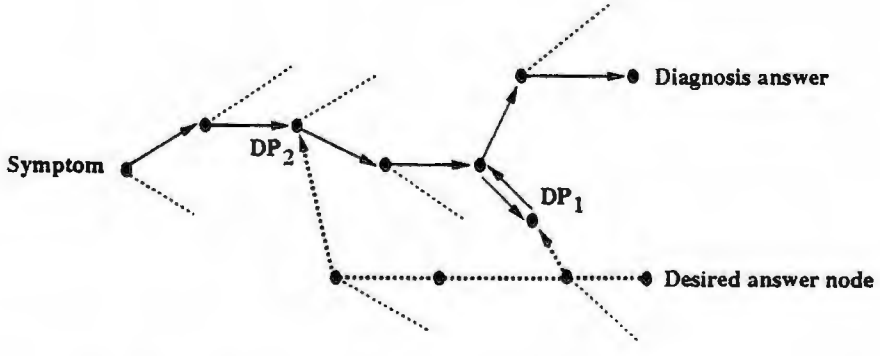


FIG. 17. Backward analysis from the desired answer node.

to a lower value than that of B" or "Change the relational weight of event E_1 to a higher value than that of event E_2"

5 Case Study

We have applied KASE to develop a defect diagnosis expert system for color picture tube manufacturing. The manufacturing of color picture tubes consists of many sub-processes. Some visible defects, such as "bald patch" or "unevenness in color", are found in the last testing process. This system aims to find the cause of many kinds of visible defects in color picture tubes and to decide appropriate countermeasures, such as "adjust the room temperature" or "shorten the exposure time".

In the system analysis phase, we had several meetings with domain experts to analyze their needs and the task domain. It was finally confirmed that the problem-solving method using the KASE tools could be applied as those which domain experts use for this task.

The system development phase began with training the domain experts in the use of KASE tools. The domain experts were beginners in programming and manipulating work-stations. However, in this training, they only needed one day to master the basic operations for the KASE tools. At the same time, the domain experts started to gather diagnostic knowledge for one kind of visible defect and to arrange it into a multi-relational network model. This took about two weeks to complete about 100 events of domain knowledge. Then, all of the fragmentary knowledge was input into DiPROS with the support of DiKAST by the domain experts themselves. This took about one more week. The knowledge engineers only consulted with domain experts about the manipulation of the KASE tools and the arrangement of the knowledge. As a whole, the development of the prototype diagnostic system could essentially be established in three weeks by the domain experts alone, who had no previous experiences in developing expert systems. Furthermore, the knowledge base maintenance can easily be accomplished with the KASE tools.

Other diagnostic system development projects are currently underway with KASE, such as system diagnosis of traffic signal equipment and defect diagnosis for casting processes.

6 Conclusions and Future Extension

We developed a diagnostic KASE and applied two KASE tools to real-world expert system development.

The KASE tools provide high level knowledge representation styles for describing diagnostic knowledge. From the view point of knowledge base visualization and maintenance ability, the multi-relational network model is appropriate for the initial development of the knowledge base, that is, to collect diagnostic knowledge from many kinds of knowledge sources and systematize it as a knowledge base. In our target problem domain, strategic knowledge plays an important role in decreasing the cost of diagnosis. Both "relational weight between related events" and "execution cost of diagnostic test" are simple but appropriate representatives for the domain experts.

DiKAST is intended to support the knowledge verification or refinement phase of knowledge base building after the initial development phase. In other words, this phase includes

compensating for the knowledge shortage, repairing redundant knowledge and modifying incorrect knowledge which leads to some erroneous solution. The KA interview includes three kinds of knowledge analysis and guides the knowledge base modification. These interview strategies were formulated by formalizing the interview manners for experienced knowledge engineers. For example, the viewpoint switching interview in the hypothesis testing knowledge analysis not only guides in compensating for missing diagnostic tests, but also plays an important role in refining domain knowledge. Furthermore, when some erroneous diagnosis occurs, DiKAST seeks the candidates for misjudgment through backward searching from the desired solution node, and proposes some knowledge modification plans in order to change the erroneous inference path to the desired direction. Compared with the existing KA tools, this inference path analysis has an advanced feature. This interview also has the ability to modify the strategic knowledge in the knowledge base.

Through several applications, we have confirmed the effectiveness of the diagnostic KASE. After short-term training, domain experts were able to input their diagnostic knowledge through the guidance of DiKAST and complete a knowledge base in a short period of time by themselves alone.

In contrast to the above, we could not apply KASE tools for some diagnosis domains because of the mismatches in the expertise model with the target problem. The diagnosis on electric power system is an example. This problem is to detect fault locations in the circuit, kinds of fault, and undesirable operations of protective relays. This problem does not satisfy one of the constraints for applying our KASE: all possible diagnosis results are enumerated in the knowledge base. In this problem, the domain expert creates the candidates for the diagnosis result in the course of diagnosis, from the behavioral knowledge regarding the circuit breaker and the topological knowledge about the power transmission circuit.

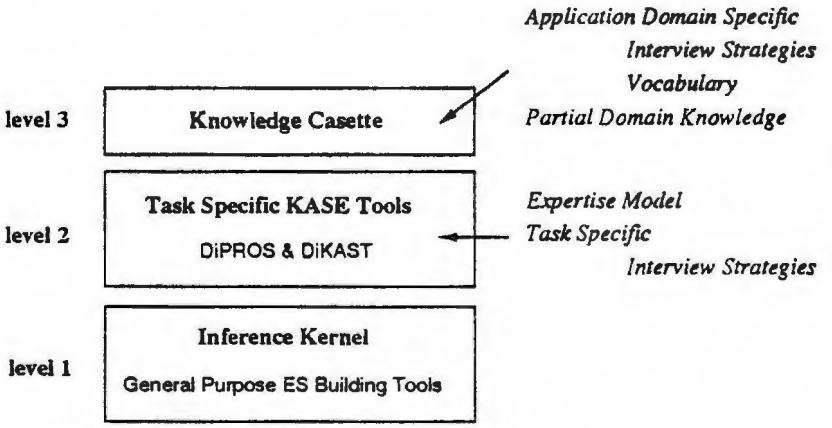


FIG. 18. Extended KASE.

We have some plans for the future extension and investigation of the next generation KASE, which consists of three hierarchical levels (→ Fig.18). The first level is called the *inference kernel*, which corresponds to the general purpose expert system building tool. The first level KASE tools are intended to be able to be applied to any kind of problem. The second level is called the *task specific KASE tool.* Both DiPROS and DiKAST were developed on the diagnostic task specific expertise model and included in this second level KASE. We

also developed a production scheduling task specific knowledge acquisition support tool, named PROKAST, in this level. Currently, we are investigating the formalism of the third level of KASE, called the domain specific *knowledge cassette*. The knowledge cassette will possess some application domain specific knowledge parts and can be used as an enhancement to the corresponding task specific KASE tools. For example, the knowledge editor and KA tool use some domain specific vocabulary, and the partially completed knowledge base will be included in this level.

Acknowledgments

We gratefully thank Seiichi Nishijima, Takeshi Kohno, and Takashi Yamaguchi of Systems & Software Engineering Laboratory, TOSHIBA Corporation, for providing essential support. We are also thankful to the technical staff of Fukaya Display Devices Works, TOSHIBA Corporation, for their valuable comments and discussion.

References

BENNET, J. S. (1985). A Knowledge-based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, *Journal of Automated Reasoning*, Vol.1, pp.49-74.

BOOSE, J. H. (1985). A Knowledge Acquisition Program for Expert Systems based on Personal Construct Psychology, *International Journal of Man-Machine Studies*, Vol.23, pp.495-525.

BOOSE, J. H., BRADSHAW, J. M. (1987). Expertise Transfer and Complex Problems : Using AQUINUS as a Knowledge-acquisition Workbench for Knowledge-based Systems, *International Journal of Man-Machine Studies*, Vol.26, No.2.

BRADLEY, J. BRIGGS, K. H. (1989). The Symptom-Component Approach to Knowledge Acquisition, *SIGART Newsletter*, Vol.108, pp.70-76.

BYLANDER, T., MITTAL, S. (1986). CSRL : A Language for Classificatory Problem Solving and Uncertainty Handling, *AI Magazine*, Vol.7, pp.66-77.

BYLANDER, T., CHANDRASEKARAN, B. (1987). Generic Tasks for Knowledge-based Reasoning : The "Right" Level of Abstraction for Knowledge Acquisition. *International Journal of Man-Machine Studies*, Vol.26, No.2.

DIEDERICH, J., LINSTER, M. (1989). Knowledge-Based Knowledge Elicitation. In G. Guida & C. Tasso (Eds.), *Topics in Expert System Design*, pp.323-350, Elsevier Science Publishers.

ESHELLMAN, L. (1988). MOLE : A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In S. Marcus (Ed.), *Automatic Knowledge Acquisition for Expert Systems*, pp.37-80. Kluwer Academic Publishers.

GRUBER, T. R., COHEN, P. R. (1987). Design for Acquisition : Principles of Knowledge-System Design to Facilitate Knowledge Acquisition. *International Journal of Man-Machine Studies*, Vol.26, No.2.

GRUBER, T. R. (1989). A Method for Acquiring Strategic Knowledge, *Knowledge Acquisition*, Vol.1, No.2.

KARBACH, W., LINSTER, M., VOSS, A. (1990). Models, Methods, Roles and Tasks: Many Labels - One Idea?. *Knowledge Acquisition*, Vol.2, 279-299.

MARCUS, S. (1988). SALT : A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In S. Marcus (Ed.), *Automatic Knowledge Acquisition for Expert Systems*, pp.81-123. Kluwer Academic Publishers.

MUSEN, M. A., FAGAN, L. M., COMBS, D. M., EDWARD, D. (1987). Use of Domain Model to Drive an Interactive Knowledge-editing Tool. *International Journal of Man-Machine Studies*, Vol.26, No.2.

MUSEN, M. A. (1989). Conceptual Models of Interactive Knowledge Acquisition Tools. *Knowledge Acquisition*, Vol.1, No.1.

WILKENS, D. C., CLANCEY, W. J., BUCHANAN, G. G. (1987). Knowledge Base Refinement by Editing Abstract Control Knowledge. *International Journal of Man-Machine Studies*, Vol.27, pp.281-294.

SOME THEORETICAL ISSUES ON THE COMPUTER INTEGRATED DOCUMENTATION PROJECT

Guy A. Boy[1]
NASA Ames Research Center
Mail Stop 244-17, Moffett Field, CA 94035, U.S.A.
Tel. (415) 604-3369
Email: boy@pluto.arc.nasa.gov

ABSTRACT

In this paper, we discuss some of the theoretical issues arising from our work on Computer Integrated Documentation (CID), as well as its relation to other work. CID's knowledge-based indexing mechanism allows situational knowledge acquisition from text by experimentation. It utilizes on-line user information requirements and suggestions either to reinforce current indexing in case of success or to generate new knowledge in case of failure. The main problems are to navigate in hypertext documentation and to acquire indexing knowledge from librarians and end-users. Management and maintenance of large documentation systems is developed. In particular, this paper proposes a semantic similarity measure to help users navigate in large documentations.

INTRODUCTION

In this paper, we discuss some of the theoretical issues arising from our work on Computer Integrated Documentation (CID). CID is a system that enables integration of various technical documents in a hypertext framework and includes an intelligent browsing system that incorporates semantic indexing in context. Semantic indexing has been investigated by several authors. Dumais et al. (1988) propose a method for organizing nodes into a semantic structure on the basis of the overlap of the descriptors used in the referents. Stotts and Furuta (1988) proposed a model of hypertext based on Petri nets. Their system enforces browsing restrictions, e.g., deactivates some hypertext links. Like CID, a medical handbook system described by Frisse and Cousins (1988) separates the "index space" from the "referent space". They have shown how some index architectures can be exploited for enhanced information retrieval, query refinement, and automated reasoning. Their index space model is based on inference using belief networks of descriptors. The I^{3}R system (Thompson and Croft, 1989) uses a Bayesian inference network, constructed from the information in a hypertext database, to answer queries. Crouch et al. (1989) have used cluster hierarchies to help navigate in hypertext structure. All these contributions to information retrieval developed methods for refining links between descriptors and referents. However, the concept of context has not been presented explicitly in any of them.

Weyer (1988) advocates the fact that information should be adaptable to the learner's preferences, and links should depend on the user's previous actions and current goals. This point of view supports perfectly our knowledge-based approach to hypertext. Other approaches have been developed to acquire and refine links on-line. For instance, Kibby

---

[1] Dr. Boy is currently at NASA under contract NAS2-13210 to Sterling Federal Systems.

and Mayes (1989), in their StrathTutor hypertext system try to eliminate the need for exclusively manual methods for creating links between hypertext nodes by generating links based on knowledge acquired when the user browses through the system. Also, Monk presents a method for constructing a personal browser (Monk, 1987). In this approach, the system monitors the user's navigation behavior and interrupts the user to ask whether it should add a node to the browser when it has been accessed frequently.

Semantic indexing in context is equivalent to the acquisition of situational knowledge from text. We claim that such a situational knowledge capture should take place during browsing and/or information retrieval. By situational (or contextual) knowledge we mean knowledge elicited from text that is purposeful in a given context. Other approaches have been developed to acquire knowledge from text (Regoczei & Hirst, 1989) using a technique called "Sortal Analysis". Mooney (1990) in GENESIS uses a preexisting domain theory to elicit new concepts by using explanations. Delisle, Matwin and Zupan (1991) have combined explanation-based learning with text analysis in order to acquire knowledge from text. Moulin and Rousseau (1990) in SACD have developed an approach to understand and acquire knowledge from prescriptive texts, i.e., already in the form IF-THEN rules. However, noone in the knowledge acquisition comunity has concentrated efforts on the way situational knowledge implicitly included in text is constructed and used, i.e., indexing knowledge.

This paper proposes an approach to navigation in hypertext and semantic indexing in context. Navigation is seen as the process of going from a document to another taking into account the current context and previous knowledge on the possible moves in similar contexts (section 1). The main problem of this approach is to acquire the appropriate descriptors of documents and their attached contextual conditions of use. Descriptors are semantic directions in the hypertext. Documents (referents) are particular locations to be visited by users. Distance between documents can be defined as semantic correlation between document description (see section 2). This paper presents these concept in the framework of CID. Section 3 presents a method for context clustering. This method is particularly useful for organizing contextual knowledge that is acquired on-line from users' feedback. In the balance of the paper, we start a discussion on possible extensions of this approach to a variety of similar domains.

1. NAVIGATION IN HYPERTEXT SPACE

1.1. Descriptors and Referents

Indexing is the process of building descriptors {d} (descriptions) from referents {r} and linking them together in a given context (r->d). We call a *descriptor* any piece of text (word, sentence, or paragraph) or image (marked area or label on part of an image) that describes objectively or subjectively any other piece of documentation. Descriptors can be single-term or multiple-terms. We call a *referent* any piece of documentation (word, line of text, paragraph, picture, moving video or animated sequence, program, volume, library) that is described by at least one descriptor. Referents are always characterized by an objective identifier, but they also can have subjective identifiers. Let r be a referent, $\{d_1, d_2, ..., d_n\}$ a set of descriptors each of them independently describing r. There must be at least one d_i that uniquely describe r. For instance, the address of this referent is such a descriptor. Generally, the other descriptors (other than d_i) are added because they are more convenient for users, even if they do not uniquely describe r. Descriptors are not necessarily included explicitly in the content (text or graphics) of a referent. A referent can be any part of the documentation including the table of contents and the index. Tables of contents or indexes are usually found more convenient because of their well known structure (hierarchical and alphabetical).

Information retrieval is the process of retrieving referents from available descriptors in a given context (d->r). Information retrieval is a process of *abduction* using indexing knowledge, i.e., knowing r->d and d, r becomes a valid hypothesis. We say that indexing defines a set of semantic relations that are used in information retrieval. Figure 1 shows an example of semantic relations between referents via descriptors.

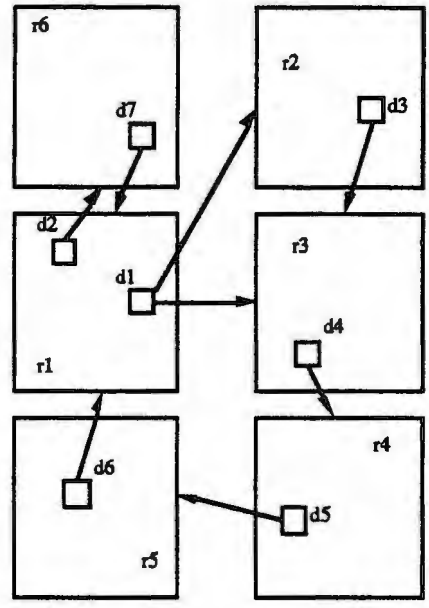


Figure 1. Example of semantic relations between referents via descriptors.

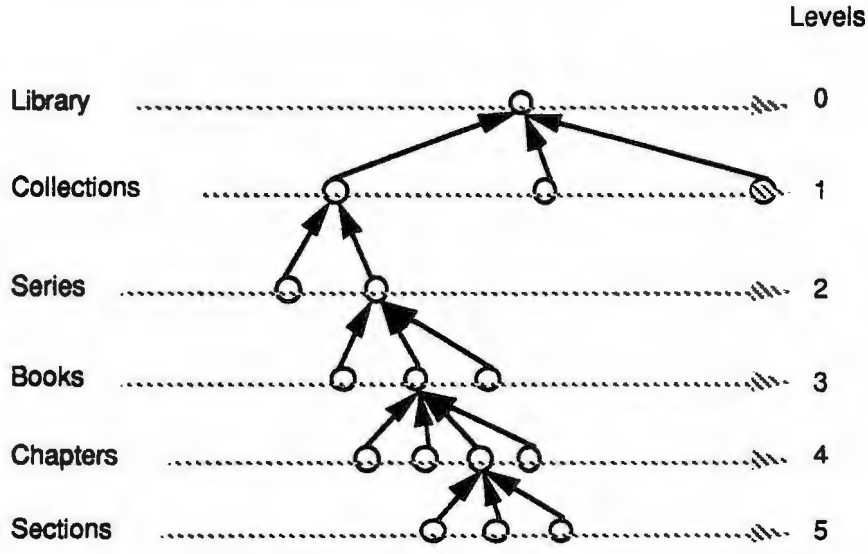


Figure 2. Example of hierarchical structure in conventional book systems (referent space).

In a conventional book, the table of contents is a list of descriptors that have a one-to-one mapping into a corresponding list of referents. As a matter of fact, the table of contents is directly a list of referents. These descriptors/referents are presented hierarchically to help

the user browse the book. As we observed that people like to use this hierarchical feature, one of the basic assumptions in CID is that its referent space is hierarchically structured. In other words, we deliberately take the metaphor of conventional books for the design of electronic documentation, i.e., books are structured hierarchically as shown in Figure 2. In CID, such a hierarchical structure is kept as the backbone of the documentation. However, transverse links can be built. In a conventional book, you also get a list of descriptors, called index, that is presented alphabetically. We also kept this metaphor in CID, i.e., each referent has its own index.

1.2. Hypertext Metalevel

Separating the knowledge-base indexing and information retrieval mechanism from the actual hypertext provides modularity of CID, i.e., both can be edited and maintained separately. However, this is not the only important consideration in this issue. Generally, indexing and information retrieval knowledge is not explicit and constitutes human expertise. If the goal is to help user browsing and retrieving appropriate information, such knowledge should be made implicit and implemented in an intelligent assistant system, as described in (Boy, 1991a), built on top of the hypertext. It is indeed the *hypertext metalevel*. A major advantage of this architecture is that this meta-level is easily programmable and allows the inclusion of knowledge on the links of the hypertext.

Furthermore, knowledge blocks have been defined and developed to represent procedures (Boy, 1989; Mathé, 1990), as well as their embedded inference and knowledge acquisition mechanisms (Boy, 1990). We have already shown (Boy, 1990) that the block representation maps well into the hypertext representation is a complementary reason to support such a separation. In particular knowledge blocks adapt well for representing procedures used in document browsing. They are used to represent links between descriptors and referents. A knowledge block contains a goal, a set of conditions, and a set of actions to achieve the goals. Conditions are decomposed into triggering conditions, abnormal conditions, and contextual conditions. In CID, triggering conditions are represented by descriptors that the user select. Contextual conditions may describe external environment, e.g., type of user, type of task when doing the consultation of documentation, time period, etc., or internal history of the current documentation use, e.g., the last visited referents, the last selected descriptors, etc. Actions are represented by referents where the user wants to go. A goal is reached when the user selects the success postcondition corresponding to the chosen referent. Similarly, an abnormal condition corresponds to a failure postcondition.

Users browsing and formulating their descriptions of the referents content contribute to the augmentation of the knowledge base by acknowledging success (or failure) postconditions after a successful (or unsuccessful) search. The more the block base grows, the more it becomes an ontology of the knowledge included in the documentation as perceived by users. The CID can then be considered as a knowledge acquisition tool for users to build their own ontology of the documentation. This overall process has been called: *index management and maintenance.*

1.3. The Block Representation as a Navigation Aid

1.3.1. Blocks as contextual links

In the documentation domain, blocks are used to represent *contextual links* between descriptors and referents. This aspect has been developed in (Boy, 1991b). In this view, documentation can be represented by two entities: referents and contextual links. As each

referent r is described by a set of descriptors D(r), a contextual link between two referents r_1 and r_2 implicitly assume that r_1 and r_2 share at least one common descriptor. Then, we can define the set of semantically common descriptors will be defined in section 2.3.

In CID, referents are represented by a database (text or graphics), and contextual links are represented by a knowledge base (contextual links) as shown in figure 3. Each descriptor in a referent is a *semantic direction* towards a set of related referents. For instance, let assume that the referent r1 has two semantic directions (descriptors d_1 and d_2) corresponding to the following contextual links[2]:

$\{d_1; (r_2 \mid C_{12}), (r_3 \mid C_{13})\}$ and
$\{d_2; (r_4 \mid C_{14}), (r_5 \mid C_{15})\}$.

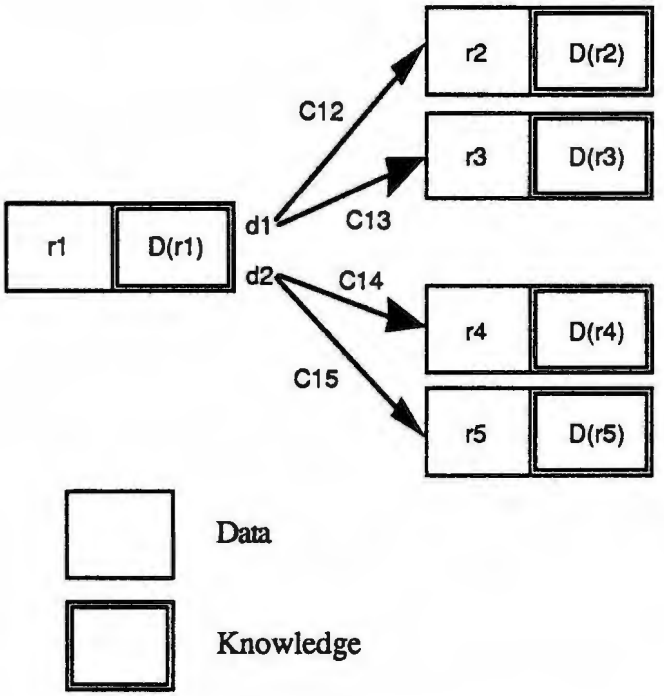


Figure 3. Descriptor-referent semantic directions.

If the current context is properly identified and the system knowledge is stabilized (i.e., sufficient contextual links have been built), then contextual links help the user to decide the current most appropriate action to take. For instance, if the user is facing the referent r1, then the system proposes two possible semantic directions d_1 and d_2, if the user selects d_1 the system proposes to go to r_2 or r_3 according to the current context. In this sense, a referent and its associated knowledge base could be called an *active document*, i.e., a *documentation agent* that has the capability of helping the user for his next move. The first major issue in this approach is the acquisition of general contextual conditions (an approach to this problem is provided in section 3 of this paper). A second important issue is to help the user finding his way in the documentation quickly. In the next subsection, we will introduce the reader to main problem of semantic correlation between referents as a navigation aid.

---

[2] $\{d_1; (r_2 \mid C_{12}), (r_3 \mid C_{13})\}$ must be read: the contextual link between the descriptor d_1 and the referent r_2, in the context C_{12}, and the referent r_3, in the context C_{13}.

1.3.2. Semantic correlation helps to decide what to do next

As we already mentioned, contextual links can help users decide what to do next. The way descriptors and referents are organized defines the referent space topology. In other words, the structure of the referent space influences the decisions of users when they browse. It is also reasonable to assume that user's ability to find the information they are looking for in the referent space depends on their ability to navigate in this environment. This navigation problem includes knowledge about the navigation capabilities as well as knowledge about the domain (i.e., documentation contents). Indeed, it is clear that if users know about the vocabulary of the domain, descriptors-referents relationships are likely to be more understandable than for novices who have to learn and eventually build their own domain ontology. We claim that presentation of an ordered list of semantically correlated referents helps the user to decide where to go next. CID can automatically compute semantic correlations between semantically neighbor referents. We call neighbor referents two referents r_1 and r_2 that share at least one descriptor.

2. ACQUISITION OF INDEXING KNOWLEDGE

2.1. Generation of Descriptors

Documentation users should be provided with a thesaurus of descriptors. This will reduce the number of new descriptors that have meanings similar to existing descriptors (defined by other users). However, if users do not find any convenient descriptor in the dictionary, they must be able to define a new one. Definition of such new descriptors should be checked for consistency with the rest of the descriptors. In practice, whenever a descriptor is generated, it must be sent up in the documentation hierarchy. This consistency check problem is detailed below.

There are two sets of levels that must be defined. First, a major assumption in CID is that the documentation is organized hierarchically. For instance, level 0 corresponds to the library level, level 1 to the collection level, and so on (see Figure 2). We will define these levels as hierarchical levels in the documentation or "doclev". Second, descriptors are also structured hierarchically. In this case, we will talk about hierarchical levels in the descriptors or "deslev". For instance, if a descriptor d_1 is defined at deslev n-1 and d_2 at deslev n, and d_1 and d_2 are dependent then d_1 is a generalization of d_2. Even if doclevs do not map onto deslevs and conversely, there is some loose connections between them. The overall thesaurus of a documentation is ratified at the highest doclev. This ratification process also provides an explanation of why a new descriptor has been assigned to a specific referent. In particular, when a large documentation is being built, people designing higher doclevs must be consulted when new descriptors are being built in lower doclevs.

In CID, as each referent has an index (i.e., a list of descriptors), an index of a referent r at doclev n of the documentation structure should be a generalization of the index of a referent that is a child of r at doclev n+1 (Figure 2). The concept of inclusion has to be understood taking into account the concept of classes and objects. For instance, a descriptor at level n may be a class of another descriptor at level n+1. Let d_1 be a descriptor at doclev n, there exists necessarily a descriptor d_2 at each doclev $m < n$ that is either d_1 or a class of d_1:

$$\begin{aligned} &\forall r_p,\ \text{doclev}(r_p)=n,\ \forall d_1 \in D(r_p),\ \forall m<n, \\ &\text{if } \exists r_q,\ \text{doclev}(r_q)=m \text{ and } r_p \text{ childOf}^{(1)}\ r_q \\ &\quad \text{then } \exists\ d_2 \in D(r_q) \text{ and } d_1 \text{ childOf}^{(1)}\ d_2 \end{aligned} \qquad (1)$$

where doclev(r) is a function that maps the set of referents onto the integer set {0, 1, 2, ...}, the relation "r_p childOf$^{(1)}$ r_q" is true when r_p is a child of r_q in the referent space (the same definition applies in the descriptor space for "d_1 childOf$^{(1)}$ d_2"), and D(r) is the set of descriptors attached to the referent r. If the expression (1) is not verified for a particular descriptor d_1 then d_1 has to be added to (or a new class d_2 has to be defined that includes d_1) the index of referents r_q that are located on the path from r_p to the root referent (level 0 in the referent space). Thus, testing the satisfaction of (1) is a mean of improving completeness and consistency of the overall dictionary.

It is interesting to note that the descriptor space and the referent space should have the same partial ordering relations between related classes in the two spaces. In particular, if the descriptor space maps the referent space well, navigation in the documentation is facilitated.

2.2. Semantic Indexing in Context

In the conventional approaches to indexing, indexing is described as a frequency-based process (Frisse & Cousins, 1989). But, indexing would be improved if users' needs could be taken into account in index generation and maintenance. Generally, users refine (and often define) their needs by trial and error evaluation of information retrieval results. Such an assumption requires an interacting environment that allows incremental acquisition of indexing knowledge.

Indexing is a decision making process that must be directly accessible to the user. This decision process concerns building semantic relations, i.e., relations between descriptors, between referents, and between descriptors and referents. The corresponding knowledge representation used to handle this decision process must allow repair in case of failure, i.e., when the user did not make the right decision. It should allow incremental transformation.

There is a tradeoff between on-line and off-line index acquisition. We claim that semantic indexing in context must be on-line. Indeed, it is very difficult, and often impossible, for a user to elicit situational knowledge off-line. Conversely, when the user is performing an information retrieval task, it is easier to elicit contextual conditions attached to links. If indexing has to be performed by the user, how much can we ask him/her without disturbing them in his/her primary task, that is, in the usual case, information retrieval. This is an open question that still has to be tested. It is actually a basic research issue to investigate the interface requirements for intelligent human-machine cooperation. Acceptance of such new technology by the user is a very complex issue. Indeed, the feedback the user receives with this technique may greatly influence acceptance.

2.2.1. Analysis of user's traces

Traces are sequences of users' actions and system responses, i.e., traces resulting from users browsing in the referent space. CID provides a record mode in which traces can be stored and replayed to elicit statistically significant patterns, such as sequences of referents that the user visit frequently according to a descriptor selection.

When users select "success", this means that they are satisfied with the information they have gotten in trace T. However, it is not possible to know from this data if the complete trace is needed or part of it. If such a trace is recorded and analyzed off-line, it would be possible to know statistically what conjunction of referents the user needs when he selects a descriptor.

A typical trace analysis follows. (1) Generally, the user selects descriptors that he/she finds close to his/her needs. However, all of them may not be relevant. The user may chose a descriptor because other potentially more relevant descriptors are not currently available.

The first problem is then to identify the relevant descriptors in the trace. (2) Some referents may provide a piece of needed information but not be sufficient for the user. This may be a reason for him/her not to select the "success" button and select another descriptor in the corresponding referent. An important question is: shall we keep the last link (d_{n-1} -> r_n), or the link (d_0 -> r_n), or an intermediate link (d_j -> r_n ($0 \le j \le n-1$)), or a chain of referents (including r_n) after d_j?

2.2.2. Incremental reinforcement from user's feedback in context

We have chosen the following heuristic to record a minimal set of parameters during the use of the system. We have noticed that the relevance of a contextual link $\{d_i, (r)\}$ between a descriptor d_i and a referent r depends on user's feedback (success or failure) on r, the frequency of feedbacks on r, and the importance that the user assigns to this referent r. The *semantic relevance* $Rel(r \mid d_i.C)$ of a referent r with respect to a descriptor d_i in context C has been formally defined as follows:

$$Rel(r \mid d_i . C) = \sum_{s \in S_{r \mid d_i.C}} \frac{I_s}{t-t_s} - \sum_{f \in F_{r \mid d_i.C}} \frac{I_f}{t-t_f} \qquad (2)$$

where t is the current time, t_s (t_f) is the time when the user selected the success s (failure f), $S_{r \mid di.C}$ ($F_{r \mid di.C}$) is the set of successes (failures) so far at time t, and I_s (I_f) is the importance assigned to r by the user with respect to d_i in context C. In practice, I_s and I_f may be set up as constants.

2.3. Semantic Correlation between Referents

In the Space Station documentation application, documents (referents) are generated by people that are not necessarily connected to each other. They are theoretically connected through a top-down hierarchical structure such as the one presented in Figure 2. However, there is no transverse link anticipated by the current documentation system. This is actually a classical problem in the construction of a very large documentation, e.g., sometimes two similar referents can be generated by independent teams. The problem is then to check if similar referents are describing the same things. If this is the case, can they be merged into a single referent?

In order to solve this problem, it is necessary to generate well-defined descriptors that can be shared by other documentation writers. If these descriptors are well defined, we can expect to check the degree of interdependence between referents by measuring their semantic correlation.

A semantic correlation between two referents should express the resemblance between the content of each referent. It can be constructed taking into account the descriptors attached to each referent. These descriptors characterize dimensions along which each referent can be located in the descriptor space. If descriptors express the semantics of the referent space, then the descriptor space can be called semantic space by extension. The main problem is that these dimensions are not independent. As already explained, the descriptor space represents relations between descriptors that express inheritance or property relations. Furthermore, it seems reasonable to let people index their referents without constraining them with the problem of descriptor dependence. The problem is then to design a representation that makes explicit descriptor dependencies. The basic idea is to build a *semantic network* where nodes are descriptors, and links are hierarchical (inheritance) or

property links. Taking this approach, descriptors are organized into *descriptor dependency clusters*. These clusters may be loosely connected or actually independent.

We consider that the following parameters affect semantic correlation between two referents r_1 and r_2:

- the number of descriptors $n_{ds}(r_1, r_2)$ shared between r_1 and r_2,
- the semantic relevance of a referent with respect to a descriptor in a given context (derived from user feedback),
- the hierarchical level of the descriptors in each referents,
- the number of properties shared by each shared descriptor[3].

Let us assume that we want to compare two referents r_1 and r_2. Salton (1989) defines the similarity between two referents r_1 and r_2 by a function of the number of descriptors shared by both referents. r_1 and r_2 have two descriptor lists $D(r_1)$ and $D(r_2)$. The intersection between $D(r_1)$ and $D(r_2)$ is computed using the following rules:

$$\text{If } d \in D(r1) \text{ and } [\exists\, d' \in D(r2), \exists\, n, \text{ such that } d' = \text{childOf}^{(n)}(d)] \text{ then } d \in D(r1) \cap D(r2) \quad (3)$$

$$\text{If } d \in D(r1) \text{ and } [\exists\, d' \in D(r2), \exists\, n, \text{ such that } d = \text{childOf}^{(n)}(d')] \text{ then } d' \in D(r1) \cap D(r2) \quad (4)$$

where the function $d'=\text{childOf}^{(n)}(d)$ means that d' is a child of generation n of d. For instance, $\text{childOf}^{(0)}$ would be identity function, $\text{childOf}^{(1)}$ would be the direct child function (first generation), $\text{childOf}^{(2)}$ would be the grand child function (second generation), etc. In other words, we keep the most general descriptors in the intersection.

If the current context is taken into account, then the descriptor list must be restricted to the list of descriptors corresponding to the contextual links that are valid in the current context.

If the set $D(r_1) \cap D(r_2)$ is not empty, then it can be ordered with respect to their corresponding semantic relevances. We obtain the following table:

$D(r_1) \cap D(r_2)$	d_1	d_2	d_3	...
r_1	$rank_{11}$	$rank_{12}$	$rank_{13}$	...
r_2	$rank_{21}$	$rank_{22}$	$rank_{23}$	...

where $rank_{12}$ is the rank of descriptor d_2 in the subset $D(r_1) \cap D(r_2)$ of the descriptors of r_1. Each $rank_{ij}$ is computed from the relative position of d_i in r_j with respect to the relevance $Rel(r_j \mid d_i.C)$. For instance, the above table shows the following order:

$$Rel(r_1 \mid d_1.C) \leq Rel(r_1 \mid d_2.C) \leq Rel(r_1 \mid d_3.C) \leq ...$$
$$Rel(r_2 \mid d_1.C) \leq Rel(r_2 \mid d_2.C) \leq Rel(r_2 \mid d_3.C) \leq ...$$

The Spearman rank correlation coefficient applied to this problem gives the following formula (Snedecor, 1946):

---

[3] This parameter is expected to be relevant and important, however we do not take it into account at this stage of the development of CID.

$$\rho_{Sem}(r_1, r_2 \mid C) = 1 - \frac{6 \sum_{d_i \in D(r_1) \cap D(r_2)} [rank_{1,i} - rank_{2,i}]^2}{n_{ds}(r_1, r_2)^3 - n_{ds}(r_1, r_2)} \quad (5)$$

where $n_{ds}(r_1, r_2)$ is the number of descriptors shared between r_1 and r_2, the following relation: $1 \leq rank_{ji} \leq n_{ds}(r_1, r_2)$ holds for j=1 or 2, and $0 \leq \rho_{Sem}(r_1, r_2 \mid C) \leq 1$.

This semantic similarity measure can be used as a navigation aid. Two generic cases can be described as derivations of ordered lists of referents semantically correlated to: a given referent, or a complex query.

1. From a referent r_1, $\rho_{Sem}(r_1, r_2 \mid C)$ may be computed for all the referents r_2 that share descriptors with r_1.

2. A complex query involving the conjunction of several descriptors can be represented by a set D(Q). An analogous formula $\rho_{Sem}(Q,r \mid C)$ can be derived for any referent r that shares at least one descriptor with Q.

In both cases, the corresponding ordered list of referents can be presented to the user as a suggestion of what to do next.

In the acquisition of contextual links, it can be used to fine tune referent descriptions. For instance, if two referents r_1 and r_2 are very much correlated but the user do not agree with this result, then referent descriptions $D(r_1)$ and/or $D(r_2)$ have to be revised.

3. CONTEXTUAL LINKS CLUSTERING

Let us assume that we have a knowledge base including knowledge blocks (contextual links) of the form described in (Boy, 1990). Each contextual link includes a context pattern π_i that is a conjunction of contextual conditions c_{ji}:

$$\pi_j = (c_{j1} \wedge c_{j2} \wedge ... \wedge c_{jn}) \quad (6)$$

The distinction between descriptors and contextual conditions is important both for knowledge acquisition and performance. Descriptors are defined to "describe" referents. A contextual condition is an explicit description of a situation in which a referent is relevant. It constitutes extra situational knowledge added to a descriptor of a referent in a given situation. It is clear that this distinction does not make descriptors context-independent. Obviously, some descriptors may implicitly include context already. If referents were perfectly described in all situations, this distinction would only be useful from the performance point of view. Whenever the link d->r is tested by the user in a context C, C may be added to it. This consequently implies that if the user query is $d \wedge C$ then r will be retrieved.

On the one hand, descriptors are not usually independent. A first simplification of the problem is to consider that some descriptors are classes of others. We do not consider descriptors as properties of others in this paper. On the other hand, we expect a large number of contextual links to be incrementally generated. We also expect regularities coming out of the context patterns. For instance, the same class of users access the same documents in similar situations. This can be handled by machine learning techniques such as learning from examples (Mitchell, 1982) or conceptual clustering (Fisher, 1987). In this section, we present clustering of contexts only.

Our approach[4] to constructing conceptual clusters of contexts is incremental, i.e., new objects need to be assimilated one at a time. In his COBWEB system, Fisher (1987) proposes operators that allow incremental incorporation of new objects (in our case context patterns) into a classification tree, where each node is a probabilistic concept that represents an object class. These operators include: classifying the object with respect to an existing class, creating a new class, combining two classes into a single class, and dividing a class into several classes. We will take the same approach to context clustering.

3.1. Placing a context pattern in an existing class

Quoting Fisher, placing a new context pattern in an existing class is probably the most natural way of updating a set of classes. Let's take an example that will illustrate the method. A new context pattern is (Paul Smith ∧ Diagnosing PGS). We already have the classification tree presented in Figure 4. Each contextual condition c_{ji} belongs to a class or is a class itself. In the current example (Figure 4), "Paul Smith" belongs to the class of "Operations People", and "Diagnosing PGS" belongs to the class "Diagnosing". Then the conjunction (Paul Smith ∧ Diagnosing PGS) is an instance of the existing class (Operations People ∧ Diagnosing).

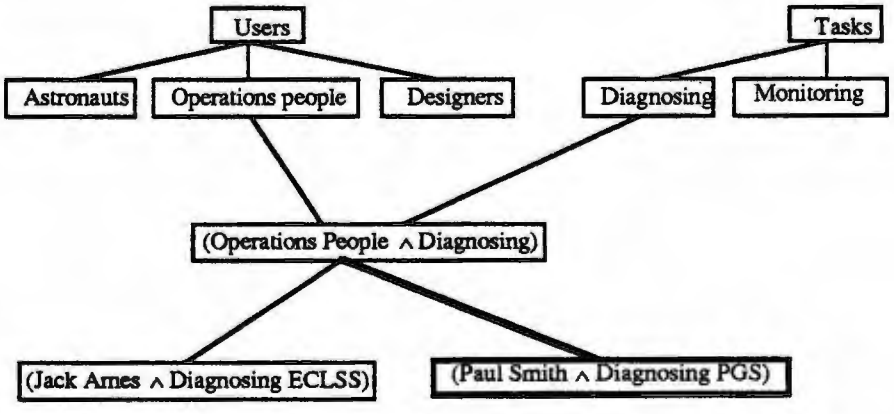


Figure 4. Placing a new context pattern in an existing class.

3.2. Creating a new class

A new class is created whenever a new context pattern cannot be placed into an existing class. This new class is identical to the new context pattern. It can be linked upwards to classes of some of the contextual conditions if any. In the current example (Figure 5), "Barry North" belongs to the class of "Designers", and "Diagnosing FTS" belongs to the class "Diagnosing". Then the conjunction (Barry North ∧ Diagnosing FTS) is an instance of the new class (Designers ∧ Diagnosing).

[4] This part of the paper is purely theoretical, we have not started any implementation yet.

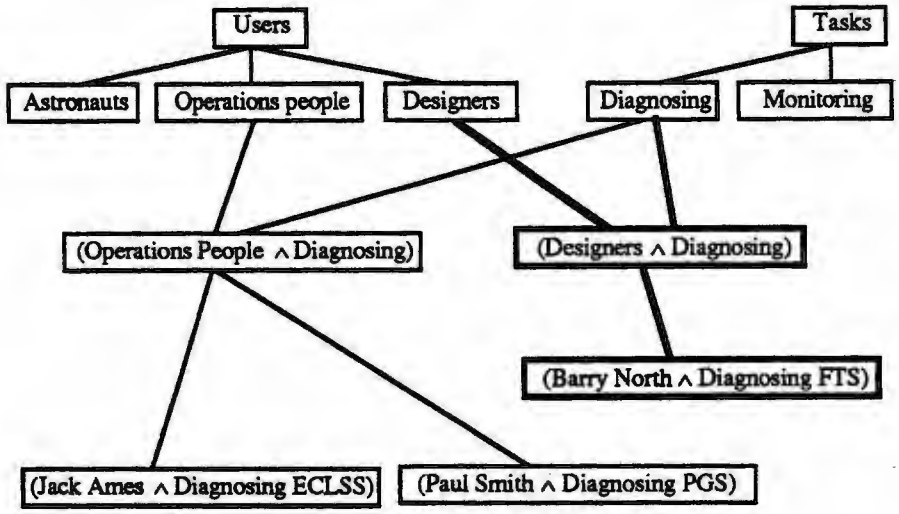


Figure 5. Creating a new class.

3.3. Merging several classes into a single class

As we assumed that compound classes are conjunctions of classes or objects, merging two classes involves creating a new class and replacing the instance-conjuncts by the corresponding class-conjuncts of the classes being merged. For instance in Figure 6a, "Operations People" and "Designers" are both "Users", and they are both performing a diagnostic task. Figure 6b shows the resulting context network after merging (Operations People ∧ Diagnosing) and (Designers ∧ Diagnosing FTS) into (Users ∧ Diagnosing).

The way we do merging can also be called generalization.

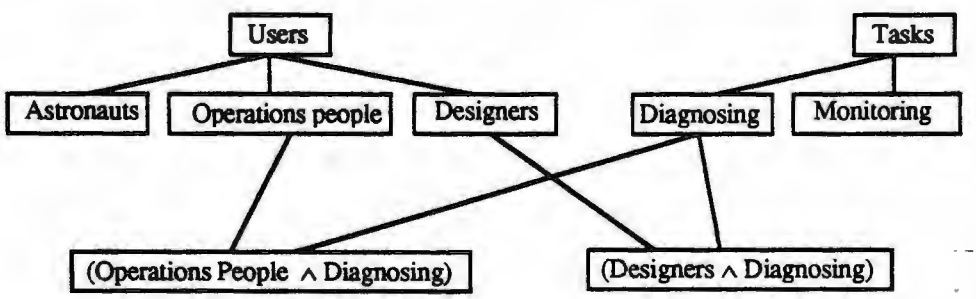


Figure 6a. Initial context classes before class merging.

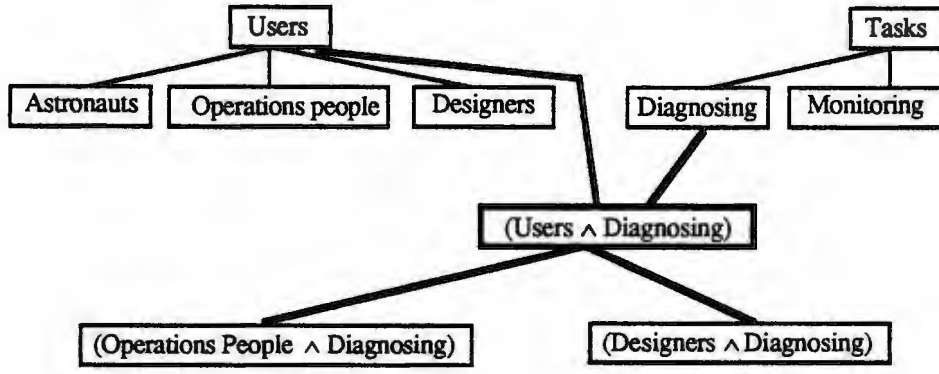


Figure 6b. Context classes after class merging.

3.4. Splitting a class into several classes

Splitting is roughly the inverse operator of merging (Fisher, 1987). Splitting happens when a compound class is an over-generalization. For instance, we may realize afterwards that (Users ∧ Diagnosing) is too general because it happens that the class (Astronauts ∧ Diagnosing) is very different from the other classes (Operations People ∧ Diagnosing) and (Designers ∧ Diagnosing). Thus, two classes will be created (Astronauts ∧ Diagnosing) and (Engineers ∧ Diagnosing) for instance (Figure 7). If the class "Engineers" does not exist already, this process leads to the creation of a new class, subclass of "Users" and superclass of "Operations People" and "Designers".

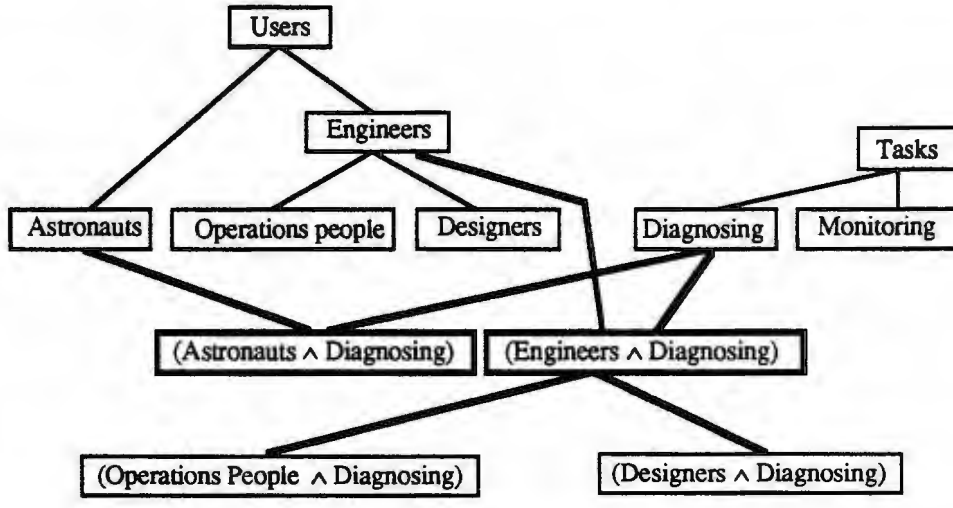


Figure 7. Splitting a class into several classes.

CONCLUSION

In this paper, we have described an approach to contextual navigation in hypertext. Even if this effort in still under development, first results show that reinforcement learning is a very appropriate tool for acquiring and refining contextual links from user's feedbacks. The methods that we are developing in the documentation domain can be easily extended to other domains such as electronic mail management, computer supported cooperative work, relational databases, or intelligent tutoring systems. Typically, this approach will be useful in domain where contextual knowledge is a key factor for efficient information retrieval, management and maintenance.

A semantic similarity measure has been proposed. It allows comparison of sets of descriptors in the current context. It can be used as a navigation aid. Two generic cases have been described as derivations of ordered lists of referents semantically correlated to: a given referent, or a complex query. In the acquisition of contextual links, it can be used to fine tune referent descriptions. It should be noted that each referent description in context represents a particular viewpoint on the corresponding referent.

Context clustering is important to organize situational knowledge (contextual links). This is useful from a knowledge acquisition viewpoint as well as a performance viewpoint. From a knowledge acquisition viewpoint, context clustering allows to build a taxonomy of contexts. From a performance viewpoint, it allows faster access to contextual links.

Finally, a contextual link knowledge base can be seen as a user model. This work is then a departure from current work on user modelling research, in which user models (possibly acquired automatically) are exploited by the system in predefined ways. As a result, these systems cannot update their user models based on experience. In contrast, our system can perform this updating automatically. In other words, our system learns to *operationalize* user models automatically, based on experience.

ACKNOWLEDGEMENTS

I would like to thank the anonymous reviewers of this paper for very pertinent comments. Philippa Gander, Nathalie Mathé and Yaron Gold also provided astute advice towards improving the quality of this paper. Special thanks to Yaron Gold for productive discussions that have shaped many of the ideas reported in this paper. I am grateful to Bharathi Raghavan, Fabian Garcia Pastor and Joshua Rabinowitz for their contributions to the CID system.

REFERENCES

Boy, G.A. (1989). The Block Representation in Knowledge Acquisition for Computer Integrated Documentation. *Proceedings Knowledge Acquisition for Knowledge-Based Systems, AAAI Workshop*, Banff, Canada.

Boy, G.A. (1990). Acquiring and Refining Indices According to Context.*Proceedings of the Fifth AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 4-9.

Boy, G.A. (1991a). *Intelligent Assistant Systems*. Academic Press, London.

Boy, G.A. (1991b). Indexing Hypertext Documents in Context.*Proceedings of the Hypertext'91 Conference*, San Antonio, Texas, December.

Croft, W.B. & Turtle, H. (1989). A Retrieval Model for Incorporating Hypertext Links. *Hypertext'89 Proceedings*, pp. 213-224, ACM press, New York.

Crouch, D.B., Crouch, C.J. & Andreas, G. (1989). The Use of Cluster Hierarchies in Hypertext Information Retrieval. *Hypertext'89 Proceedings*, pp. 225-237, ACM press, New York.

Delisle, S., Matwin, S. and Zupan, L. (1991). Explanation-Based Learning Helps Acquire Knowledge from Natural Language Texts. *Proceedings of the Sixth International Symposium on Methodologies for Intelligent Systems*, Charlotte.

Dumais, S.T., Furnas, G.W., Landauer, T.K., Deerwester, S. and Harshman, R. (1988). Using Latent Semantic Indexing to Improve Access to Textual Information. *Proceedings of the ACM CHI'88*, pp. 281-285, Washington D.C., May 15-19.

Fisher, D.H. (1987). Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning*, 2, pp. 139-172.

Frisse, M.E. and Cousins, S.E. (1989). Information Retrieval from Hypertext: Update on the Dynamic Medical Handbook Project. *Hypertext'89 Proceedings*, pp. 199-212, ACM press, New York.

Kibby, M.R. & Mayes, J.T. (1989). Towards Intelligent Hypertext. In McAleese, R. (Ed.) *Hypertext Theory into Practice*, Albex 1989, pp. 164-172.

Mathé, N. (1990). Assistance Intelligente au Controle de Processus: Application à la Télémanipulation Spatiale. PhD Thesis, ENSAE, Toulouse, France.

Mitchell, T.M. (1982). Generalization as Search. *Artificial Intelligence*, 18, pp. 203-226.

Monk, A. (1989). The personal browser: A tool for directed navigation in hypertext systems. *Interacting with Computers*, 1, 2, pp. 190-196.

Mooney, R.J. (1990). A General Explanation-Based Learning Mechanism and its Applications to Narrative Understanding. Morgan Kaufmann Publishers.

Moulin, B. and Rousseau, D. (1990). A Knowledge Acquisition System for Analyzing Prescriptive Texts. *Proceedings of the 4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, pp. 22.1-22.19.

Regoczei, S. and Hirst, G. (1989). Sortal Analysis with SORTAL, a Software Assistant for Knowledge Acquisition. *Proceedings of the 4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, pp. 25.1-25.18.

Salton, G. (1989). *Automatic Text Processing. The Transformation, Analysis, and Retrieval of Information by Computer*. Addison Wesley, Reading.

Snedecor, G.W. (1946). *Statistical Methods*. Fourth Edition. The Iowa State College Press, Ames, Iowa.

Stotts, P.D. and Furuta, R. (1988). Adding Browsing Semantics to the Hypertext Model. *Proceedings of the ACM Conference on Document Processing Systems*, pp. 43-50, Santa Fe, NM, December 5-9.

Thompson, R. and Croft, W.B. (1989). Support for Browsing in an Intelligent Text Retrieval System. *International Journal of Man-Machine Studies*, 30, pp. 639-668.

Weyer, S.A. As we May Learn. In Aubron, S. & Hooper, K. (1988). *Interactive Multimedia: Visions of Multimedia for Developers, Educators, & Information Providers*. pp. 87-103, Microsoft Press.

Knowledge Representation for Knowledge Acquisition: A Three-Schemata Approach

Jeffrey M. Bradshaw
Research & Technology, Boeing Computer Services
P.O. Box 24346, M/S 7L-64, Seattle, Washington 98124; (206) 865-3422;
jbrad@atc.boeing.com

Kenneth M. Ford
Institute for Human and Machine Cognition
University of West Florida, Pensacola, Florida 32514; (904) 474-2551; kford@ai.uwf.edu

Jack Adams-Webber
Department of Psychology
Brock University, St. Catharines, Ontario, Canada L2S 3A1; (416) 688-5550, ext.3714

Knowledge acquisition is a constructive modeling process, not simply a matter of "expertise transfer." Consistent with this perspective, we advocate knowledge acquisition practices and tools that facilitate active collaboration between expert and knowledge engineer, that exploit a serviceable theory in their application, and that support knowledge-based system development from a life cycle perspective. Many of the serious problems associated with knowledge acquisition and maintenance stem directly from the inadequacies of the particular knowledge representations used at a given stage of knowledge base refinement. To counter these problems, we propose an approach to knowledge representation for knowledge acquisition that distinguishes between external, conceptual, and internal schemata. As an implementation of the external schemata, we emphasize the use of *mediating representations* which serve as a means of communication between expert and knowledge engineer. *Intermediate representations* implement the conceptual schema, and help bridge the gap between the mediating representations and a particular implementation formalism. ICONKAT and DDUCKS are briefly described as examples of constructivist knowledge acquisition tools embracing the modeling perspective and built around collections of mediating and intermediate representations. We conclude with a discussion of the issues in design and evaluation of mediating representations.

Advances in knowledge acquisition theory and practice have led to a better understanding of knowledge representation requirements for effective knowledge acquisition. To aid an appreciation of these requirements, we will first consider the importance of a modeling perspective for knowledge acquisition (Section 1). Then we will discuss the role of mediating and intermediate knowledge representations in facilitating the modeling process, and present a three-schemata approach for knowledge representation that we have found useful in our work (Sections 2-3). We will describe ICONKAT and DDUCKS as examples of knowledge acquisition tools embracing the modeling perspective and built around collections of mediating and intermediate representations (Sections 4-5). Finally, we will discuss issues in the design and evaluation of mediating representations (Section 6).

1. Model-based Knowledge Acquisition

Recent work in knowledge acquisition has emphasized that the creation of knowledge bases is a constructive modeling process, and not simply a matter of "expertise transfer" (e.g., Bradshaw &

Boose, 1990; Clancey, 1990; Cox, 1991; Ford, Bradshaw & Adams-Webber, 1991; Gruber, 1990; Musen, 1988; Shaw & Woodward, 1990; Wielinga, Akkermans, Schreiber & Balder, 1989). More fundamentally, modeling is purposive, that is, to be involved in modeling is necessarily to be engaged in using the model in some particular setting for particular reasons that together determine what should be modeled, how to model it, and what can be ignored (Thimbleby, 1990; Winograd & Flores, 1986). Together, the criteria of purpose and cost-effectiveness determine how additional pragmatic issues should be resolved such as who the users of the model are, how it ought to be presented in order to be both usable and useful, and how it will be maintained over its projected lifetime (Rothenberg, 1989). This modeling perspective implies the need for active participation by both experts and knowledge engineers in the creation of knowledge bases.

The view of knowledge acquisition as a modeling activity has discredited the faulty notion that the process consists of "mining those jewels of knowledge out of [the experts'] heads one by one" (Feigenbaum & McCorduck, 1983, p.2). Such a perspective erroneously assumes that there exists some 'gold standard' of knowledge and that a domain expert is one who has captured a discrete (presumably large) part of the 'reality' underlying observed events in the relevant domain. This 'mining analogy' is both fundamentally at odds with cognitive science theory (see e.g., Agnew & Brown, 1989), and dangerously misleading as a metaphor for the guidance of knowledge acquisition practice. Expertise is not a natural resource that can be extracted, harvested, transferred, or captured. Experts involved in knowledge acquisition are not restating a coherent body of knowledge that already exists in their minds; rather they are engaged in a constructive modeling process, in the context of which formal representations are newly created and shaped (Clancey, 1989; Ford, Bradshaw & Adams-Webber, 1991).

From a constructivist perspective, a model is not a 'picture' of the problem, but rather a device for the attainment or formulation of knowledge about it (Kaplan, 1963). Often, the most important outcome of a specific knowledge acquisition project is neither the resulting knowledge-based system, nor the solutions it provides, but rather the insights gained through the process of articulating, structuring, and critically evaluating the underlying model (Moore & Agogino, 1987). From this, we infer that the value of the knowledge acquisition effort may derive not so much from a final 'correct' representation of the problem, as from our success in framing the activity as a self-correcting enterprise that can subject any part of the model to critical scrutiny, including our background assumptions. From this standpoint, the crucial question for knowledge engineers is not "How do we know the model is correct?" (every model is an incorrect oversimplification); but rather "How useful is the model (and the modeling process) as a means of facilitating our understanding of the domain?"

This understanding of models and the modeling process entails a life cycle perspective on knowledge acquisition. Knowledge acquisition does not culminate at some arbitrary point in development, but rather extends throughout the life of the knowledge-based system. The ideal knowledge acquisition tool would support the gradual evolution of the model through numerous cycles of refinement.

Each phase of knowledge base refinement imposes its own requirements and difficulties. Many of the serious problems associated with knowledge acquisition and maintenance stem directly from the inadequacies of the particular knowledge representations used at a given stage of development. Most knowledge acquisition tools are quite limited in both their repertoire of representational devices for modeling as well as their support for representational evolution and transformation. The ideal knowledge acquisition tool would support a smooth transition of the model from an easily communicated, relatively unconstrained statement of the problem to an unambiguous specification of design for the performance system. A number of changes in representation may be required to accompany successive stages in model construction: from mental models to increasingly refined conceptual models via elicitation and analysis techniques,

and eventually, from these highly elaborated models to an operational knowledge base via formalization and implementation procedures (Shaw & Woodward, 1989; Shaw, Woodward & Gaines, 1990).

Unfortunately, the emphasis given to rapid prototyping in traditional accounts of knowledge acquisition, along with the faulty notion that 'the production of working code is the most important result of work done', often leads to the premature encoding of knowledge in an implementation formalism associated with a specific performance environment (Bradshaw & Boose, 1989; figure 1). The unfortunate result is that no independent statement of the knowledge will exist other than the rule base and possibly some glossaries in the help information of the system (Johnson, 1989).

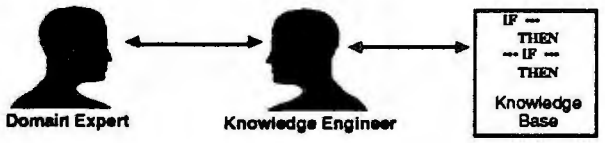


Figure 1. A traditional depiction of the knowledge acquisition process.

The problems of premature encoding of knowledge in implementation-driven representations have spurred efforts to develop other representations that more adequately support the early stages of conceptual modeling. We call these *mediating representations*.

2. The Role of Mediating Representations

In the most general sense, we can think of a representation as being a set of conventions for describing some aspect of the world (Winston, 1984). For example we can think of alphabets, mathematical symbols, musical notation, and engineering drawings as all being different forms of representation, each fit for some specific purpose.

There are many different logically equivalent methods by which the same information can be represented. Traditionally, the choice of a knowledge representation formalism has been based upon considerations of representational adequacy, inferential adequacy, and inferential efficiency (Rich, 1983). Much of the work in knowledge representation has focused on these problems. In contrast, our work is focused on the *acquisitional efficiency* of alternative knowledge representations.

Wielinga, Akkermans, Schreiber & Balder (1989) present a number of compelling arguments for making a clear distinction between knowledge-level conceptual models and implementation-focused design models in the knowledge-based system development process (see also Schreiber, Akkermans & Wielinga, 1990). Sowa (1987) has likewise argued that considerations of human efficiency far outweigh considerations of machine efficiency for complex modeling problems. He observes that, since the time used for transformation from one knowledge representation to another is linearly proportional to the length of the formula, a change of notation increases efficiency by constant factor:

> "But if a problem is truly NP hard, a change of notation will not make it solvable in polynomial time. Yet, because one notation for logic can always be translated into another, it is possible to do the input and output in a form that has the best human factors, and than translate it to the form that has the fastest theorem prover. Although the issues of machine efficiency are not irrelevant, for knowledge representation and acquisition, they are less important than naturalness and ease of use..." (see also Sowa, 1991)

The term *mediating representation* has various interpretations in the literature, however we take it to "convey the sense of... coming to understand through the representation" (Johnson, 1989, p. 184). A crucial feature is that mediating representations should be "easily readable by those who were not involved in the original development programme..." (Diaper, 1989, p. 34). This is essential, since executable knowledge bases are seldom organized for direct use by humans, but instead for the convenience of the reasoning mechanisms of the performance environment. The design of a mediating representation, on the other hand, should be optimized for human understanding rather than machine efficiency.

The choice of a representation can have an enormous impact on human problem solving performance (e.g., Larkin & Simon, 1987). As an example, consider that numbers may be represented as Arabic numerals, Roman numerals, or as bits in computer memory. While each of these forms may be logically equivalent, they are not the same in a practical sense. For example, it is much more efficient for a computer to multiply numbers represented directly as bits than to convert them from numeric symbols. Similarly, from a human point of view, it is easier to do multiplication problems with Arabic numerals than with Roman numerals.

Work on mediating representations aims to improve the knowledge acquisition process by developing and improving representational devices available to the expert and knowledge engineer (Ford, Bradshaw & Adams-Webber, 1991). A mediating representation provides an explicit external medium in which experts can model their difficult-to-articulate knowledge (figure 2). Whatever structure is inherent in the form of representation provides grounding constraints that both facilitate expression within the set of conventions adopted, and enforce comprehensibility and consistency by preempting certain possibilities. The mutual development of permanent "cognitive artifact" supplementing the exchange of information between participants promotes and enriches communication, leading gradually to a shared understanding of the emerging conceptual model of the domain (Norman, 1988, 1991). In this way, mediating representations enable domain experts and knowledge engineers to cooperatively build problem solving models. In the later stages of system development, mediating representations may also facilitate maintenance and explanation by enabling both knowledge engineers and the people eventually using the system to explore the conceptual domain model without resorting to low-level representations (e.g., C code, Lisp, rules).

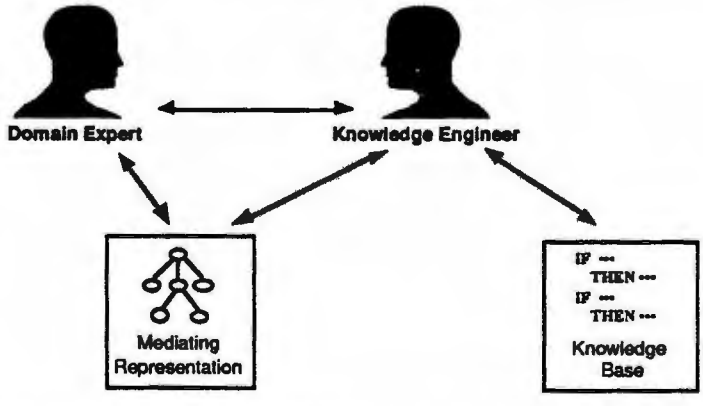


Figure 2. Mediating representations facilitate communication and understanding between domain expert and knowledge engineer.

A number of automated knowledge acquisition tools have incorporated effective mediating representations. These tools tend to adopt one of two approaches. Either they contain interfaces that bear a close resemblance in appearance and procedure to the original manual task—for example, cancer-therapy protocol forms in OPAL (Musen, 1988) and engineering notebooks in

vmacs (Sivard, Zweben, Cannon, Lakin & Leifer, 1989), or they rely on some easily-learned, generic knowledge representation form—for example, repertory grids and directed graphs (Boose & Bradshaw, 1987; Eisenstadt, Domingue, Rajan & Motta, 1990; Ford, Stahl, Adams-Webber, Cañas, Novak & Jones, 1991; Gaines & Shaw, 1986a)

3. The Role of Intermediate Representations

Since knowledge acquisition, like all modeling activities, is a process of iterative refinement, we would like to be able to map back and forth from the kinds of representations used in performance environments to mediating representations that are more useful for communication purposes. For example, initial approaches to knowledge acquisition in ETS (Boose, 1984, 1986a), KSS0 (Gaines & Shaw, 1986b), and *Nicod* (Ford, 1987; Ford, Petry, Adams-Webber & Chang, 1991) embodied procedures for transformation from repertory grids to rules. This was found to be a useful and productive step for knowledge engineers, particularly in the early prototyping phases of a project. Some kinds of information, however, could not be conveniently represented in simple repertory grids. Furthermore, this was essentially a one-way procedure—while the kinds of knowledge available in repertory grids could be transformed to rule sets, in most cases there was no natural mapping from rules back to grids.

Over time the semantic gap between acquisition systems and performance systems has widened dramatically. A distinguishing characteristic of some of the newer tools is the degree to which they promote the use of multiple perspectives on the same information and exemplify the push toward informal textual, graphical, and multimedia forms of knowledge representation (Boy, 1989; Eisenstadt, Domingue, Rajan & Motta, 1990; Gaines & Boose, 1991). As new mediating representations have increased the richness, complexity, and subtlety of the knowledge elicited by automated knowledge acquisition tools, a requirement has emerged for *intermediate representations* that can integrate diverse perspectives and help bridge the gulf between human participants and the implementation formalism required by the performance environment. In addition, intermediate representations facilitate the integration of knowledge acquisition and performance systems, allowing rapid feedback, dynamic analysis, and verification throughout the process of system development (e.g., Shema & Boose, 1988; Linster & Gaines, 1990).

Figure 3 outlines a three-schemata architecture for knowledge acquisition tools, with mediating representations serving as external schemata, the intermediate representation corresponding to a conceptual schema, and the knowledge base as an internal schema. Obvious similarities will be seen between our suggested architecture for knowledge acquisition tools and the proposed ANSI-SPARC three-schema model for data management. The definitions for the three schemata given by van Griethuysen and King (1985) provide a good summary of this perspective:

> "The... *conceptual schema* controls what is described in the information base. The conceptual schema controls the semantic meaning of all representations, that is, defines the set of checking, generating, and deducing procedures of the information at the conceptual level in the information system.
>
> The *external schemata* describe how the users wish to have the information represented. The external processor interfaces directly with the users and coordinates their information exchange.
>
> The *internal schema* describes the internal physical representation of the information... The mapping between the external schemata and the internal schema must preserve meaning as defined by the conceptual schema."

This approach allows views containing mediating representations to be coupled to the underlying intermediate representation so that any changes made to one view may be immediately reflected in all related views. Knowledge analysis and performance tools may be similarly designed to exploit the integration of information at the intermediate level.

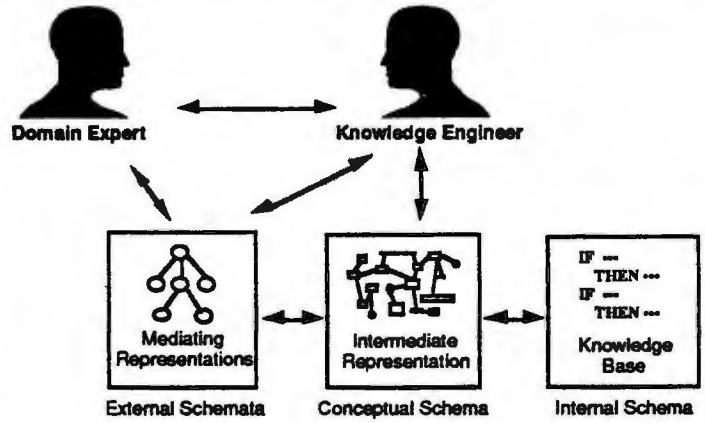


Figure 3. Three-schemata architecture for knowledge acquisition tools

An intermediate knowledge representation is one "which only exists between flanking representations and is bound to them by clearly defined projection rules which map one representation to the next" (Johnson, 1989, p. 184). The issue of mapping between representations is a troublesome one. For one thing, it is obvious that much of what can be modeled in mediating representations cannot be directly incorporated into the current commercial performance systems. Furthermore, since every transformation of knowledge is a reconstruction of that knowledge, we know that, even if logical equivalence as part of representational mapping is assured, we cannot assume practical or even conceptual equivalence. For these and other reasons, automated mapping between representations will continue to be an issue, and some amount of manual mapping from one representation to another will remain common practice. Whether mapping is automatically assisted or manual, informal or formal, does not obviate the need for integrative, intermediate representations that are relatively independent of the constraints of the delivery environment.

ICONKAT and DDUCKS are two examples of a newer generation of constructivist knowledge acquisition tools embracing the modeling perspective and built around collections of mediating and intermediate representations (Bradshaw, Ford, Adams-Webber & Boose, 1991). We will introduce these tools in sections 4 and 5, focusing our discussion mainly on modeling and representational issues.

4. ICONKAT: An Integrated Constructivist Knowledge Acquisition Tool

ICONKAT is the focus of a continuing research program aimed at the development of an integrated constructivist knowledge acquisition system incorporating principles and techniques from both Kelly's (1955) personal construct theory and Ausubel, Novak,and Hanesian's (1978) assimilation theory. It is essentially a system for interactively assisting the domain expert and knowledge engineer in cooperatively building a model of the former's expertise. On the assumption that knowledge acquisition should continue throughout the life of any expert system, ICONKAT is designed to fully support all of the facets of this process described above. Accordingly, it includes a modeling environment (Section 5.1) and three major subsystems: elicitation (Section 5.2), maintenance (Section 5.3), and explanation (Section 5.4). We will briefly describe the principal components of this tool (see Figure 4), and then explicate how they

can be used to facilitate the design, construction, testing, maintenance and explanation specific of knowledge bases.

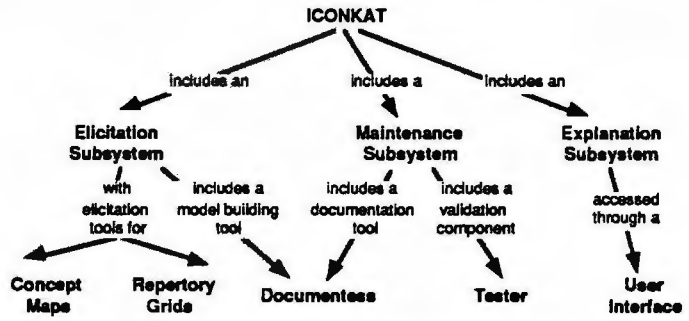


Figure 4. A concept map representation of ICONKAT's system architecture.

4.1. Representations: Mediating and Intermediate

ICONKAT's modeling environment consists of both mediating and intermediate representations, and contains all the various informational components of the knowledge acquisition process. The principal mediating representations of ICONKAT's elicitation subsystem are the concept map (assimilation theory's major methodological tool; c.f. Novak and Gowin, 1984; Ford, Stahl, Adams-Webber, Cañas, Novak & Jones, 1991) and the repertory grid (personal construct theory's major methodological tool).

ICONKAT's intermediate representation (sometimes referred to as the 'glue') consists of a collection of modeling primitives implemented as abstract data types in C++. These modeling primitives are largely the product of the expert's interaction with the mediating representations (concept maps and repertory grids) found in ICONKAT's elicitation subsystem. In addition, software tools (e.g., Documentess) are provided that facilitate the addition of supporting objects and primitives (e.g., images, audio, video, documents) to the model. Thus, the conceptual domain model is constructed within the framework afforded by the modeling primitives in conjunction with the mediating representations—providing direct support for model creation, documentation, maintenance, knowledge base generation, and the subsequent expert system's explanation facility.

Documentess (Documentation Tool for Expert Systems) is an interactive 'intelligent' documentation tool for constructing and maintaining a domain model from which associated knowledge bases can be derived. By exploiting the flexibility offered by hypermedia in structuring information, Documentess provides a single integrated interface to the conceptual domain model. Essentially, Documentess furnishes a mechanism for hierarchically structuring a relatively unstructured collection of abstract data types (i.e., the modeling primitives). It is through the Documentess interface that these comparatively labile and amorphous data structures comprising the intermediate representation are molded into a coherent domain model. Thus, it enables the underlying knowledge base to be viewed through the prism of a model largely constructed by the expert, and later reconstructed by the knowledge engineer.

For example, Documentess enables the user to shift levels of abstraction (up or down) from the elicited repertory grids, to the resulting rules expressed in a pseudo-natural language, to the actual rules formulated in the required syntactic formalism of the performance environment, and back. Documentess also associates the concepts found in the previously elicited concept maps with their related objects and classes as represented in the intermediate representation.

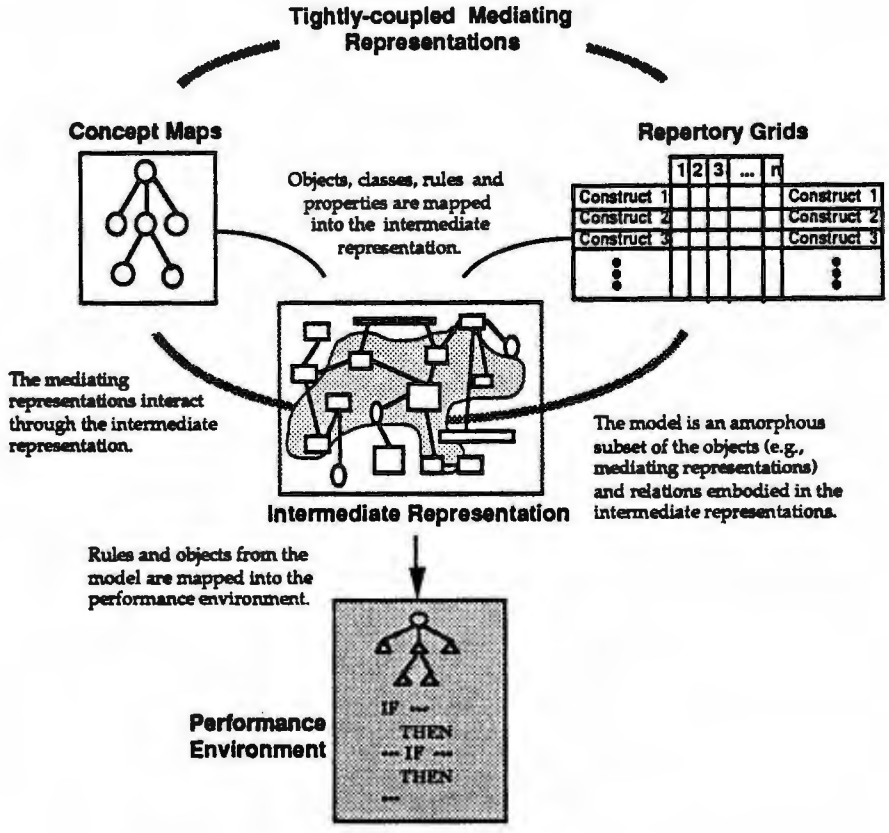


Figure 5. ICONCAT elicitation subsystem overview.

In summary, an appropriate hypermedia system integrated with an intermediate representation (the glue) enables the knowledge engineer to move smoothly between the methodological tools (i.e., mediating representations) employed in the knowledge acquisition process and the resulting rules and objects which comprise the knowledge base of the expert system under development.

4.2. The Elicitation Subsystem

ICONKAT's knowledge elicitation subsystem interactively assists domain experts in the task of building a model of their expertise. Specifically, ICONKAT interacts with the domain expert largely through the mediating representations furnished by its concept mapping and repertory grid interfaces (figure 5). It employs these complementary mediating representations in a synergistic fashion. Concept maps offer the knowledge engineer a conceptual overview of the expert's domain model, thus providing an essential initial structuring of the domain.

The components of ICONKAT's elicitation subsystem are integrated on three levels. First, and most importantly, the principal components of ICONKAT's elicitation subsystem are integrated at the theoretical level. We (as tool-makers) have exploited the epistemological overlap between personal construct psychology and assimilation theory, thus enabling repertory grids and concept maps to be integrated in a principled manner. Second, in the ICONKAT system, repertory grids and concept maps are integrated at the level of practice (domain level). Specifically, concept maps furnish a hierarchically ordered, conceptual overview of the expert's model of the domain. Therefore, concept maps can provide knowledge landscapes (essentially topographical maps) of the domain that inform the knowledge engineer about the potential and appropriate use of grids

within this domain. Third, the mediating representations (concept maps and repertory grids) of ICONKAT's elicitation subsystem are integrated at the representation level. They utilize the representation facilities (i.e., the modeling primitives or glue) of ICONKAT's modeling environment as the material basis for the construction of conceptual domain models.

4.3. The Maintenance Subsystem

On the assumption that knowledge acquisition is an ongoing process (rather than a discrete phase), ICONKAT includes a maintenance subsystem. A knowledge-based system typically evolves over its lifetime in a largely *ad hoc* fashion—often leading to a 'runaway' expert system, in which the actual knowledge base slides ever further out of conformance with the documentation, effectively rendering the knowledge base unmaintainable.

In contrast, ICONKAT's maintenance subsystem provides support tools for use by the knowledge engineering team while testing the system's performance, refining the knowledge base, and maintaining the overall system. The maintenance subsystem fully exploits ICONKAT's mediating representations (e.g., concept maps, repertory grids) to furnish various perspectives of the evolving domain model as embodied in the modeling primitives.

4.4. Explanation Subsystem

A machine that incorporates expert judgment in a given domain is more likely to find acceptance by those seeking its advice if it can explain its recommendations (Scott, Clancey, Davis & Shortliffe, 1984). Accordingly, an explanation capability should enable a user to get a complete, understandable answer to any sort of relevant question about the knowledge explicitly and *implicitly* embodied in a system's implementation formalism (e.g., rules, frames, or whatever). Nonetheless, the capacity of most current expert systems to explain their findings (i.e., conclusions) is limited to inadequate, causal descriptions of the behavior of the performance environment's reasoning mechanism.

Key to the design of explanation subsystems capable of more than shallow and/or mechanistic accounts is to recognize that the development of an explanation facility is a fundamental part of the knowledge acquisition process. Instead of arduously constructing a model of human expertise, and then 'throwing it away' (upon translation to the syntax of the performance environment), an explanation facility should exploit the mediating and intermediate representations formed during the knowledge acquisition process.

In the ICONKAT framework, the model resulting from the knowledge elicitation process is exported from the development environment to the delivery environment where it serves as the foundation of the explanation capability for the deployed system (figure 6). Hence, in addition to its roles in elicitation, documentation, and maintenance, ICONKAT's mediating and intermediate representations offer a flexible embodiment of the domain model for subsequent exploitation by the explanation subsystem. Because the explanation subsystem affords a domain model view of the knowledge base, explanations can be provided at a variety of conceptual levels, assisting the user in better understanding the domain and subsequently in elaborating his own model based on the expert's model.

In response to a user-initiated request for explanation, the performance environment is interrupted and the user is switched into the context-sensitive explanation subsystem and conveyed to an appropriate location within the multi-dimensional space representing the model. From here, the user plays an active role in the process of constructing his own explanation by freely exploring the conceptual model and selecting from among a wealth of supporting objects (e.g., audio, video, documents, images, repertory grids, concept maps, rules, etc.). The navigation problem is somewhat ameliorated by use of the top-level concept map as a guide to traversing the

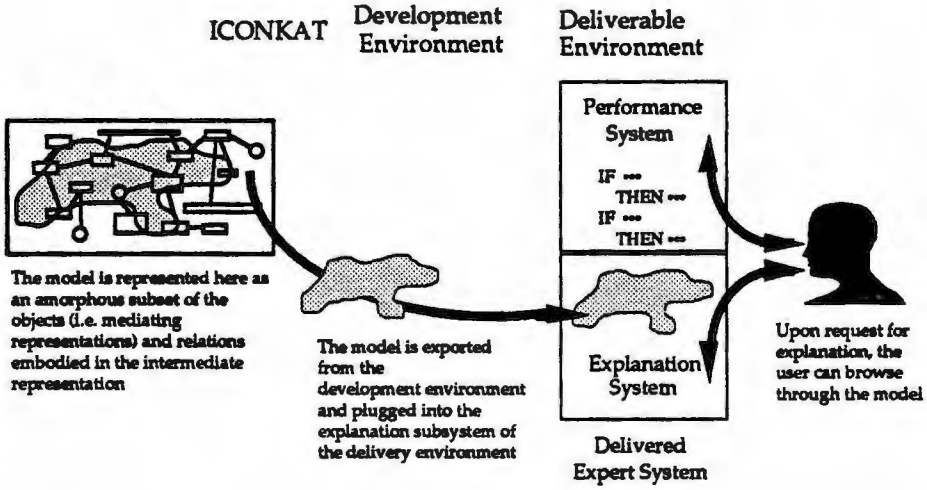


Figure 6. The transition of the domain model from the development to the delivery environment.

logical linkages among clusters of related objects. As may be expected, the user will cease browsing when he feels that he has constructed an adequate explanation from the information available.

4.5. An Application

ICONKAT is currently employed in the design and construction of NUCES: Nuclear Cardiology Expert System (Ford, Cañas, Andrews, Schad, Bezdek, Brennan & Coffey, 1991). This is a large-scale expert system for the diagnosis of first pass cardiac functional images, a non-invasive radionuclide technique used to evaluate heart wall motion abnormalities.

5. DDUCKS: Decision and Design Utilities for Comprehensive Knowledge Support

Early knowledge-based tools generally made strong assumptions about their operating environment. At first, dedicated, stand-alone applications were the rule. Over time, as the value of adding "hooks" for access to external applications and data was realized, most knowledge-based tools still operated under the assumption that they were in ultimate control of the system as the highest level executive. Currently, the greatest potential for use of knowledge-based systems is in areas requiring close interaction with traditional software applications and data. An application that assumes it is in ultimate control will be ineffective in such environments.

Brodie (1989) has discussed the need for intelligent interoperability in information systems. He defines the term to mean intelligent cooperation among systems to optimally achieve specified goals. While there is little disagreement that future computing environments will consist of multiple heterogeneous software systems running on multiple heterogeneous machines, most current computer systems are disjoint: they do not communicate.

To facilitate intelligent interoperability between knowledge acquisition tools and other software, we are in the process of implementing an "open architecture" integrating environment that allows for a high degree of connectivity among *Axotl* and other knowledge-based components (figure 7; Bradshaw, Covington, Russo & Boose, 1990, 1991; Bradshaw, Holm, Kipersztok, Nguyen, Russo & Boose, 1991). This environment is called DDUCKS (Decision and Design Utilities for

Comprehensive Knowledge Support)1.. Previously, individual components of DDUCKS have been applied to prototype a number of specific applications. As part of a Boeing project called Design of Information Systems or DIS (Benda, 1990), we are exploring how knowledge acquisition and decision support tools can work cooperatively with one another and in conjunction with commercial applications such as spreadsheets, databases, or hypermedia software, within a computer-supported meeting environment (Boose, Bradshaw, Koszarek & Shema, 1991).

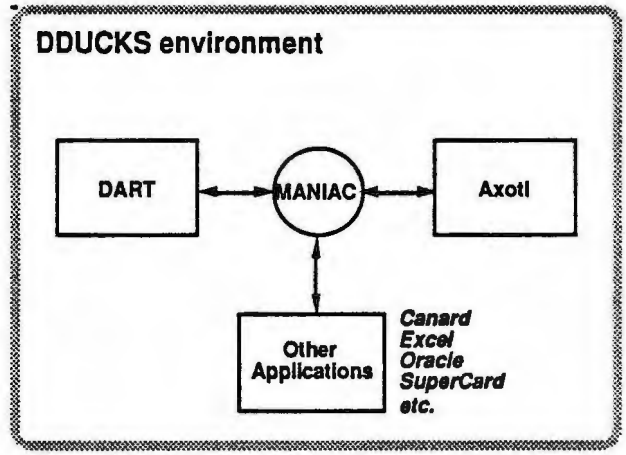


Figure 7. DDUCKS integrates components from *Axotl*, DART, and other applications.

DART. DART (Design Alternatives Rationale Tradeoffs) is a repertory-grid-based knowledge acquisition tool. It was originally developed with funding from NASA as part of an effort to capture design knowledge for the Space Station Freedom program (Boose, Shema & Bradshaw, 1990). Similar tools and concepts have been under development at The Boeing Company for many years (Boose, 1984; Boose & Bradshaw, 1987; Boose, Bradshaw, Kitto & Shema, 1991; Boose, Shema & Bradshaw, 1989). DART contains a number of elicitation, analysis, representation, and inference methods derived from personal construct theory (Kelly, 1955).

Canard. *Canard* is a knowledge acquisition tool that can be used to generate and structure complex alternatives (Bradshaw, Boose, Covington & Russo, 1989; Shema, Bradshaw, Covington, and Boose, 1990; Bradshaw, Shema, & Boose, 1991). Links are maintained between the tables and underlying repertory grids. The possibility table representation is based on the manually developed strategy tables (McNamee & Celona, 1987) and morphological charts (Zwicky, 1969) that have been used by decision analysts and designers for many years. *Canard* automates this representation and extends its logic and structure to allow knowledge-based inference and the representation of more complex problems (e.g., hierarchical tables, explicit representation of constraints). An initial prototype of *Canard* has been implemented; an effort to build a fully-featured version is now underway.

Axotl. *Axotl* combines a influence-diagram-based decision analysis workbench with knowledge-based tools to assist individuals consulting with decisions involving high stakes, difficult tradeoffs, or critical uncertainties and risks (Bradshaw, Covington, Russo, & Boose, 1990, 1991). The knowledge-based tools in *Axotl* can be configured with application-independent knowledge (i.e., knowledge of decision analysis tools and methodology) and application-specific knowledge (i.e., knowledge about a particular domain) to provide guidance and help during a consultation. An additional set of tools (which go by the name of *eQuality)* assist users in modeling and executing the decision making process (Bradshaw, Kipersztok, Holm Nguyen, Russo & Covington, 1991), and in rigorously modeling and analyzing the entities and processes involved

[1] Either the first or second *D* in *DDUCKS* is silent, depending on whether one is using it for decisions or design.

in the enterprise of which the decision is a part (Bradshaw, Holm, Kipersztok, Nguyen & Covington, 1991). The relationship between personal construct theory and decision theory is discussed in Bradshaw & Boose (1990).

MANIAC. MANIAC (MANager for InterApplication Communication) supports asynchronous and synchronous communication between any number of multitasking applications. Plans for coordination among applications are modeled and executed using knowledge-based capabilities in *Axotl*, while MANIAC provides the infrastructure for the actual message passing. Originally implemented as a driver in the 6.x version of the Macintosh operating system, MANIAC is currently being updated to take advantage of new interapplication communication features in version 7.0. Future versions may support additional hardware and software platforms.

In the following section, we will explain how DDUCKS provides an architecture for reusability (Section 5.1). Next, we will discuss mediating and intermediate representations used in DDUCKS, and an organization tool called the *project notebook* (Sections 5.2 and 5.3). Finally, we will briefly describe applications of DDUCKS within The Boeing Company (Section 5.4).

5.1. An Architecture for Reusability of Tools and Models

Because the implementation of automated knowledge acquisition tools is labor intensive, their development can often be justified only if it can be demonstrated that they can be easily applied to more than a single application. Knowledge acquisition tool developers interested in deriving the most benefit from their tools may look for areas consisting of several problems that can each be characterized by a general task model (Boose, 1989). Knowledge acquisition tools that fit the general task model can then be tailored to fit a number of specific problems.

Besides the reuse of problem solving methods, a number of researchers have also emphasized the importance of a common ontology in achieving sharable, reusable knowledge bases (e.g., Gruber, 1991; Lenat & Guha, 1990; Skuce, 1991a). Alexander, Freiling, Shulman, Rehfuss, and Messick (1988) introduced ontological analysis as a knowledge acquisition technique for the preliminary analysis of a problem-solving domain. We are currently integrating the tools in DDUCKS with the CODE version 4 system, developed by Skuce (1991b, c), as a means of defining, managing, and documenting the ontological commitments being made during the design and use of the system.

Musen (1988) was one of the first to present an explicit, general approach to creating tailorable knowledge acquisition tools. Knowledge acquisition tools are tailored through the use of a meta-level knowledge acquisition tool to edit a domain-independent conceptual model (see also Puerta, Egar, Tu & Musen, 1991). The meta-level tool, PROTOGE, provides a system to generate knowledge editors tailored for various classes of treatment plans. Physician experts can then use the knowledge editors created by PROTEGE to develop knowledge bases (e.g., OPAL) that encode specific treatment plans in their medical specialty; the resulting systems (e.g., ONCOCIN) could then be used in turn by attending physicians to obtain therapy recommendations for a particular patient.

In DDUCKS, we have tried to exploit the concept of reusability in similar way. It is useful to think of DDUCKS in terms of four layers of functionality: workbench, shell, application, and consultation (figure 8). The workbench consists of four major elements: problem-solving methods (e.g., heuristic classification, maximization of expected utility across decision alternatives, hierarchical constraint satisfaction using extended AND-OR graphs, and process optimization through event-based simulation), generic mediating representations (e.g., graph view, matrix view), a methodology-independent ontology (e.g., generic object types such as entity, attribute), and methodology-building process knowledge (i.e., knowledge about how to

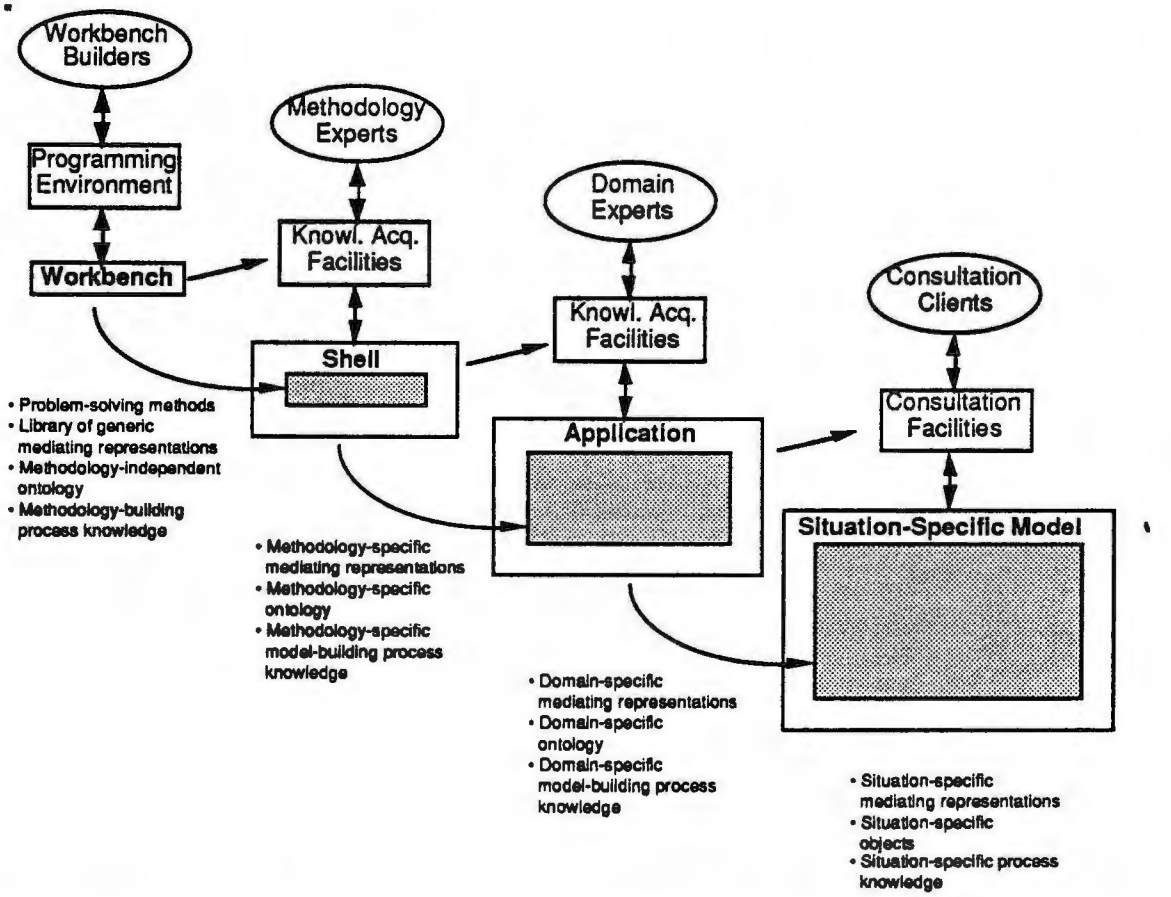


Figure 8. Four layers of functionality facilitate reusability in DDUCKS (inspired by figure from Musen, 1988).

acquire methodology-specific knowledge such as knowledge about the decision analysis process or personal-construct-based modeling process).

An instance of a shell, created by using the knowledge acquisition facilities generated by the workbench, may contain methodology-specific mediating representations created out of the generic ones (e.g., influence diagrams, repertory grids), a methodology-specific ontology (e.g., decision and chance nodes; elements and constructs), and methodology-specific model-building process knowledge (i.e., knowledge about how to acquire domain-specific knowledge such as R&D project investment decision knowledge or space station design trade study knowledge). Currently, Smalltalk-80 methods are written to implement mappings between generic and methodology-specific behavior of views, and between methodology-specific aspects of mediating representations and the intermediate representation; eventually, we hope to have an interface at this level that can be used by non-programmers.

An instance of an application, created by using the knowledge acquisition facilities generated by the shell, may contain domain-specific mediating representations (e.g., form-filling interfaces tailored to R&D investment decision makers or space station designers that may be used in place of influence diagrams or grids), a domain-specific ontology (e.g., go/no-go investment decision nodes, technical risk chance nodes; alternatives and criteria), and domain-specific model-building process knowledge (i.e., knowledge about how to conduct a consultation with clients such as R&D investment decision makers or space station designers).

An instance of a consultation, created by using the consultation facilities generated by the application, may contain situation-specific mediating representations (e.g., text and graphical annotation of the model), a situation-specific ontology (e.g., decision and chance nodes for a particular project decision model; alternatives and criteria for a particular design decision), and situation-specific process knowledge (e.g., a process model for a particular business or decision - making process).

The complete situation-specific model represents the unique characteristics of a particular problem and comprises all the information mentioned above. This model is formulated, evaluated, and analyzed during the consultation to produce recommendations for action or for further model refinement.

5.2. Mediating and Intermediate Representations

User-interface management systems (UIMS) are becoming an essential part of interactive tool development and end-user tailoring (Hix, 1990). We are extending the capabilities of a Smalltalk-80-based direct-manipulation user-interface builder to form the foundation for future versions of the DDUCKS interface (Laland, Novotny, Enzer & Bortz, 1991).

The tools in DDUCKS rely on the Smalltalk-80 MVC (model-view-controller) concept for managing consistency among views (Goldberg, 1990; Krasner & Pope, 1988; Adams, 1988b). The MVC approach provides a way to effectively factor out the data in an underlying model from the data in dependent views, so that changes to the model in one view are immediately reflected in all related views. Class hierarchy mechanisms in Smalltalk-80 allow generic views of a certain sort to be easily specialized for different purposes. This, in conjunction with the DDUCKS UIMS, has allowed us to define many different views on similar aspects of the model, as well as several similar views on different aspects of the model.

Figure 9 is a view of knowledge representation in DDUCKS. The most important form of intermediate representation in DDUCKS consists of entities, attributes, relationships and behavior. This maps very straightforwardly onto the conceptual modeling capability provided in CODE version 4 (Skuce, 1991b, c). The six views surrounding the intermediate representation correspond to the generic mediating representation building blocks that are implemented as abstract "pluggable" view classes (Krasner & Pope, 1988; Adams, 1988a) in Smalltalk-80. These representations are generic in the sense that they define the graphical form for the knowledge, but the form has no underlying semantics. Within DDUCKS, these representation can be called up as *sketchpad views* to record free-form graphical and textual information. For example, team members can capture back-of-the-envelope drawings, agendas, issues, action items, requirements, and other information pertinent to their task. Although not part of the formal model, this information can be linked to the model as a hypertext-like form of annotation.

By combining these generic mediating representations with a semantics defined in methodology-specific knowledge within the intermediate representation and (for some representations) the problem-solving method, methodology-specific mediating representations are defined. For example, influence diagrams combine a graph view with the concepts of decision, chance, and value nodes and the problem-solving method of maximization of expected utility across decision alternatives. Trade study matrices (a domain-specific kind of repertory grid) are built out of a matrix view, the concepts of alternatives, criteria, and ratings; and a heuristic classification problem-solving method. We call the representations with an underlying semantics *model views*. By virtue of the MVC paradigm, consistency is continuously maintained for all model views that rely on the same version of the intermediate representation.

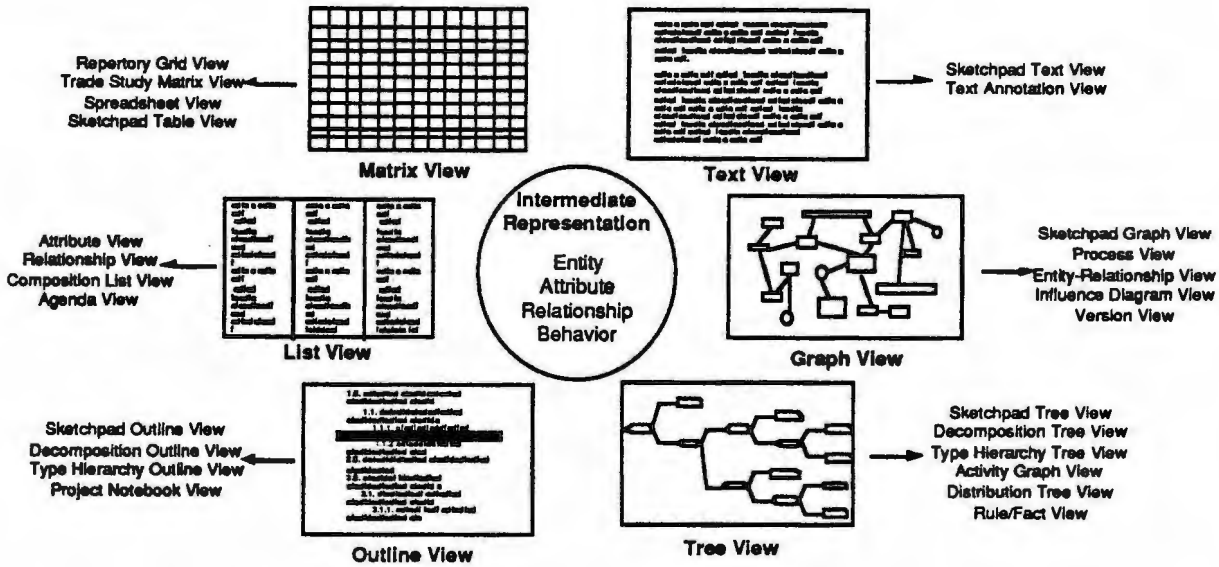


Figure 9. Intermediate representations, generic mediating representations, and methodology-specific mediating representations in DDUCKS.

5.3. Project Notebook

A *project notebook* facility helps groups of users collect and organize the diverse materials associated with a particular project and to manage changes between different versions and views of the model as it evolves. The project notebook can assist in planning and modeling activities throughout the life cycle of the project. Using project notebook *templates*, groups can configure the contents of the boiler plate project notebook to be consistent with their own organizational standards (e.g., required entity and icon types, reporting forms) and procedures (e.g., required steps in a project plan). This is analogous to pre-loading to a real notebook with labeled dividers and forms. In addition to its obvious use in managing views and models, the project notebook supports the group as a simple computer-supported meeting facilitation tool and as a form of group memory.

5.4. Applications

Various combinations of DDUCKS components have been evaluated with respect to several different kinds of problems. These have included:

- rapid prototyping for expert systems (Boose, 1984; 1986a);
- analysis and conflict resolution of knowledge from multiple experts (Boose, 1986b);
- knowledge acquisition and delivery of knowledge-based systems for heuristic classification problems (Boose & Bradshaw, 1987; Boose, Shema & Bradshaw, 1989);
- acquiring and verifying control knowledge for a blackboard system (Baum, Shema, Boose & Bradshaw, 1989);
- evaluation of training course effectiveness (Schuler, Russo, Boose, & Bradshaw, 1990);
- assisting R&D managers to make project investment decisions involving substantial uncertainty, risk, and complex tradeoffs (Bradshaw, Covington, Russo & Boose, 1990, 1991);

- design knowledge capture for a Corporate Memory Facility for NASA's Space Station Freedom (Boose, Bradshaw, Shema & Covington, 1989; Boose, Shema & Bradshaw, 1990; Bradshaw, Boose & Shema, 1991);
- alternative generation and constraint management for synthesis and design problems (Bradshaw, Boose, Covington & Russo, 1989; Shema, Bradshaw, Covington & Boose, 1990; Bradshaw, Shema & Boose, 1991); and
- documentation and streamlining of business processes (Bradshaw, Holm, Kipersztok, Nguyen & Covington, 1991).

As part of the DIS project, we are implementing enhancements to these individual tools, and developing and evaluating a methodology for their joint use in an electronic meeting room setting (Boose, Bradshaw, Koszarek & Shema, 1991).

6. Discussion: Design and Evaluation of Mediating Representations

Some of the most important unresolved issues about mediating representations concern how they should be designed and evaluated (Tortora, 1990). Most of the past work in this vein has been guided by intuition rather than principle, and evaluated by anecdote rather than empirical analysis. While some amount of this is unavoidable (and in fact desirable), we must do more to develop a theory-based mediating representation design methodology (see e.g., Carroll, Kellogg & Rosson's (1991) discussion of the *task-artifact cycle)*. Casner and Larkin (1989), for example, have begun to apply recent research in how representation affects problem solving to guide the definition of a principled methodology for designing effective perceptual codes and interpretations that support particular kinds of tasks. Criteria, derived from Johnson (1989, p. 185) and Winston (1984) also suggest general rules-of-thumb for evaluating the effectiveness of a representation:

- Is the formalism sufficiently expressive?
- Does the formalism aid communication between the members of the development team?
- Does the formalism actually guide knowledge analysis in a significant way?
- Does it make the important things explicit, suppressing detail and keeping rarely used information out of sight, but still available when necessary?
- Does it expose natural constraints?
- Is it complete and concise, efficiently saying all that needs to be said?

If we could design a representation that performed well with respect to these 'acquirability' criteria and was also endowed with Turing-equivalent 'computational expressiveness', we would have achieved our ideal. Unfortunately, as we have discussed elsewhere (Bradshaw, Covington, Russo & Boose, 1991; Ford, Bradshaw & Adams-Webber, 1991), there seems to be an inevitable tradeoff between the acquirability and computational expressiveness of knowledge representations (see also Gruber, 1989; Webster, 1988).

We expect that graphical, diagrammatic representations will prove superior to prose for tasks involving complex spatial or conceptual relationships (Larkin & Simon, 1987). As Checkland (1979) has noted:

> "a diagram is an improvement on linear prose as a means of describing connections and relationships. Looking at a map, for example, we can take it as a whole. Our minds can process different parts of it simultaneously, in parallel, whereas prose has to be processed serially, putting a much greater burden on memory if our concern is with relationships. In addition, and presumably because of this possibility of 'parallel processing,' diagrams are

automatically summaries. Imagine the amount of prose needed to convey all the information contained in a 1:50,000 Ordnance Map."

Various kinds of graphical representations will differ in their effectiveness for a given situation. For example, we find both repertory grid and directed graph representations to be useful as visual problem clarifiers. The grid presentation allows the person to see patterns of similarity and difference that would otherwise be difficult to grasp. However, dependencies, abstraction or subsumption relationships cannot readily be shown within a grid, and are most naturally displayed in directed graphs (Pearl & Verma, 1987; Pearl, Geiger, & Verma, 1988). Our experience in using both grids and directed graphs as mediating representations for knowledge convince us that each has advantages, depending on the context. As Jones (1981) states:

> "Matrices and nets are complementary ways of expressing a single set of relationships. The matrix enables a pattern that is too complex for the brain to generate all at once to be built up piece-by-piece outside the brain. A net of the same connections permits the assimilation of this pattern, once it has been completed and checked, back into the brain from whence came its constituent parts. Thus the brain can use an external aid to discover patterns among pieces of information that were originally understood only in isolation."

Beyond these general considerations, there is the fact that a mediating representation that is expressive and understandable for one set of participants may not be useful for another, either because the content is foreign to their concerns or the form in which the information is presented is unfamiliar. Zachman's work (1987), for example, has emphasized the importance of selecting or creating the appropriate representations to support different kinds of participants in information system development. He uses the definition of classical architecture deliverables and adapts them for information system deliverables. These deliverables become the cognitive artifacts to support each kind of participant in information system development. Hence, in the process of constructing a building the deliverables are as follows:

- first, the architect and the owner derive a mutual understanding of the basic concepts of the building through use of the bubble charts;
- then the architect and owner agree on the form of the building as rendered in the architect's drawing;
- next, the architect's detailed drawings establish a basis for negotiation with the general contractor;
- following this, the contractor's plans are formulated to describe the final building as seen by the builder;
- finally, the shop plans provide a subcontractor's specifications and instructions for construction of a part or section of the building.

Similar levels of deliverables characterize the idealized development of information systems (Rich and Waters, 1987):

- At the top of the hierarchy are corporate executives who formulate vision, mission, goals, objectives, plans, and product deliverables for the company in relatively brief statements of policy.
- These statements of policy are used by strategic planners and experts on specific aspects of the business to arrive at detailed enterprise-wide descriptions of business entities and processes as they are and as they should be. These descriptions are couched in the vocabulary of the business and may serve to generate requirements not only for information systems, but also for other changes such as organizational restructuring.
- Systems analysts and designers develop information system models as the basis for design of the architecture of the program, translating high-level requirements into a

detailed specification. A key feature of this specification is that it is couched in the vocabulary of programming rather than business.
- Finally, the programmer creates code in a programming language based on the detailed specification.

The important principle to draw from this illustration is that each "representation" of the building or information system has its own sphere of usefulness as an explicit basis of communication and agreement between adjacent participants in the process, and no single representation serves the needs of all. The goal is to support the most natural kind of formalization possible for a given set of participants, making the representation an effective vehicle for debugging the model. We conclude with the words of David Parnas on traditional software specification, which apply equally well to knowledge acquisition:

> "The word 'formal' has been commandeered by a bunch of people who feel that it isn't formal if human beings can read it... I have fallen into the same trap. I could write something and I could read it but my students couldn't. And they could write something and they could read it but I couldn't. And, not only that, but neither of us *wanted* to read it. ... Therefore I have worked on new ways to write specifications so that people could read it... You can't imagine how overjoyed I was when a pilot told me we had made a mistake with the A7 [avionics software specified in an earlier project] — not because we made a mistake but because the pilot could tell us." (Parnas, 1991; see also Brooks, 1987; Kapor, 1991)

It is our hope that a continued discussion and work on the design and use of mediating representations will increase the focus of the knowledge acquisition community on the importance of designing effective mechanisms for communication and shared understanding between participants in knowledge-based system development.

Acknowledgements

We express our appreciation to Mike Anderson, Miroslav Benda, Guy Boy, Kathleen Bradshaw, John Brennan, Alberto Cañas, Richard Chapman, Beverly Clark, John Coffey, Jim Fulton, Pete and Cindy Holm, Earl Hunt, William Jones, Oscar and Sharon Kipersztok, Cathy Kitto, Joe Koszarek, Johnny Liseth, Allen Matsumoto, John Miyamoto, Thom Nguyen, Steve Poltrock, Peter Russo, Bob Schnebele, Doug Schuler, Kish Sharma, Dave Shema, Howard Stahl, Steve Tanimoto, Bruce Wilson, and Jeff Yerkes for their contributions and support. This work has benefitted from ongoing discussions with Neil Agnew, John Boose, Brian Gaines, Max Henrion, and Mildred Shaw, as well as numerous colleagues in the knowledge acquisition community. *Axotl*, *Canard*, DART, *eQuality*, and DDUCKS have been developed at the Boeing Computer Services Computer Science organization. ICONKAT was developed in the Institute for Human and Machine Cognition at The University of West Florida. The design and development of ICONKAT has been supported in part by Florida High Technology Enhancement Funds.

References

Adams, S.S. (1988b). MetaMethods: Active values. *HOOPLA!*, 1(1), 3-6.

Adams, S.S. (1988b). MetaMethods: The MVC paradigm. *HOOPLA!*, 1(4), July, 5-6, 13-21.

Agnew, N.M. & Brown, J.L. (1989). Foundations for a theory of knowing: I. Construing reality. *Canadian Psychology*, 30, 152-167.

Alexander, J.H., Freiling, M.J., Shulman, S.J., Rehfuss, S. & Messick, S.L. (1988). Ontological analysis: An ongoing experiment. In J.H. Boose & B.R. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*. London: Academic Press.

Ausubel, D.P., Novak, J.D. & Hanesian, H. (1978). *Educational Psychology: A Cognitive View* (2nd edition), New York: Holt, Rinehart and Winston; reprinted, New York: Warbel and Peck, 1986.

Baum, L.S., Shema, D.B., Boose, J.H., Bradshaw, J.M. (1989). Acquiring and verifying control knowledge for a blackboard system, *Proceedings of the Fourth Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, 1989

Benda, M. (1990). Design of information systems: Towards an engineering discipline. Boeing Computer Services Technical Report. Seattle, WA: Boeing Computer Services, Computer Science Organization.

Boose, J.H. (1984). Personal construct theory and the transfer of human expertise. *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX.

Boose, J.H. (1986a). *Expertise Transfer for Expert System Design*. New York: Elsevier.

Boose, J.H. (1986b). Rapid acquisition and combination of knowledge from multiple experts in the same domain. *Future Computing Systems Journal*, 1, 191-216.

Boose, J.H. (1989). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition Journal*, 1, 3-37.

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: Using *Aquinas* as a knowledge-acquisition workbench for knowledge-based systems. *International Journal of Man-Machine Studies*, 26, 3-28. Also in J. Boose & B. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*. London: Academic Press, pp. 39-64.

Boose, J.H., Bradshaw, J.M., Kitto, C.M., & Shema, D.B. (1991). From ETS to *Aquinas*: Six years of knowledge acquisition tool development. *Knowledge Acquisition Journal*, in press.

Boose, J.H., Bradshaw, J.M., Koszarek, J.L. & Shema, D.B. (1991). Better group decisions: Using knowledge acquisition techniques to build richer decision models. Manuscript submitted to the *1992 Hawaii International Conference on Systems Sciences*.

Boose, J.H., Bradshaw, J.M., Shema, D.B., Covington, S.P. (1989). Design knowledge capture for a corporate memory facility, *IJCAI-89 Workshop on Knowledge Acquisition: Practical Tools and Techniques*, Detroit, Aug., pp. 5-6.

Boose, J.H., Shema, D.S., & Bradshaw, J.M. (1989). Recent progress in *Aquinas*: A knowledge acquisition workbench. *Knowledge Acquisition Journal*, 1, 185-214.

Boose, J.H., Shema, D.S., & Bradshaw, J.M. (1990). Capturing design knowledge for engineering trade studies. In B. Wielinga, J. Boose, B. Gaines, G. Schreiber, & M. Van Someren (Eds.), *Current Trends in Knowledge Acquisition*. Amsterdam: IOS Press.

Boy, G.A. (1989). The block representation in knowledge acquisition for computer integrated documentation. *Proceedings of the Fourth Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, October 1-6.

Bradshaw, J. M. & Boose, J.H. (1989). Knowledge acquisition as CASE for knowledge-based systems. Presentation at the *Third International Workshop on Computer-Aided Software Engineering* (CASE-89), London, July.

Bradshaw, J. M. & Boose, J.H. (1990). Decision analysis techniques for knowledge acquisition: Combining information and preferences using *Aquinas* and *Axotl*. *International Journal of Man-Machine Studies*, 32, 121-186. Also in J.H. Boose & B.R. Gaines (Eds.), *Progress in Knowledge Acquisition for Knowledge-Based Systems*. London: Academic Press.

Bradshaw, J.M., Boose, J.H. & Shema, D.B. (1991). A knowledge acquisition approach to design rationale. In J. Carroll and T. Moran (Eds.), *Design Rationale,* in press.

Bradshaw, J.M., Boose, J.H., Covington, S.P., & Russo, P.J. (1989). How to do with grids what people say you can't: The application of decision analysis methods in *Axotl* and personal construct methods in *Aquinas* to design problems. *Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, Nov., 1988.

Bradshaw, J.M., Covington, S., Russo, P., & Boose, J.H. (1990). Knowledge acquisition techniques for intelligent decision systems: Integrating *Axotl* and *Aquinas* in DDUCKS. In M. Henrion, R.D. Shachter, L.N. Kanal, & J.F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 5*. North-Holland: Elsevier.

Bradshaw, J.M., Covington, S.P., Russo, P.J. & Boose, J.H. (1991). Knowledge acquisition techniques for decision analysis using *Axotl* and *Aquinas*. *Knowledge Acquisition Journal*, 3(1), 49-77.

Bradshaw, J.M., Ford, K.M., Adams-Webber, J.R. & Boose, J.H. (1991). A constructivist approach to knowledge acquisition tool development. In K. Ford & J. Bradshaw (Eds.), special knowledge acquisition issue of the *International Journal of Intelligent Systems,* in press.

Bradshaw, J.M., Holm, P., Kipersztok, O., Nguyen, T. & Covington, S. (1991). Acquiring knowledge for process management using *eQuality*. Manuscript submitted to the 1992 *Hawaii International Conference on Systems Sciences.*

Bradshaw, J.M., Holm, P., Kipersztok, O. Nguyen, T., Russo, P. & Boose, J.H. (1991). Intelligent interoperability in DDUCKS. Presentation at the *AAAI-91 Cooperation Among Heterogeneous Intelligent Systems Workshop*. Anaheim, CA, July. Revised manuscript in preparation for publication available from the authors.

Bradshaw, J.M., Kipersztok, O., Holm, P., Nguyen, T., Russo, P. & Covington, S. (1991). Knowledge acquisition for decision analysis process knowledge. Presentation at the *AAAI-91 Knowledge-based Construction of Probabilistic and Decision Models Workshop*. Anaheim, CA, July. Revised manuscript in preparation for publication available from the authors.

Bradshaw, J.M., Shema, D. & Boose, J.H. (1991). *Canard:* An alternative generation tool based on possibility tables. In S. Kim (Ed.), *Creativity: Models, Methods, and Tools,* AAAI Press, in press.

Brodie, M.L. (1989). Future intelligent information systems: AI and database technologies working together. In J. Mylopoulos & M.L. Brodie (Eds.) *Readings in Artificial Intelligence and Databases*. San Mateo, CA: Morgan Kaufmann.

Brooks, F. P., Jr. (1987). No silver bullet: Essence and accidents of software engineering. *Computer*, April, 10-18.

Carroll, J.M., Kellogg, W.A. & Rosson, M.B. (1991). The task-artifact cycle. In J.M. Carroll (Ed.), *Designing Interaction: Psychology at the Human-Computer Interface*. New York: Cambridge University Press.

Casner, S.M. & Larkin, J.H. (1989). Cognitive efficiency considerations for good graphic design. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. August, 275-282.

Checkland, P.B. (1979). Techniques in "Soft" systems practice: Part I. Systems diagrams—some tentative guidelines. *Journal of Applied Systems Analysis*, 6, 33-40.

Clancey, W.J. (1989). The frame of reference problem in cognitive modeling. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Ann Arbor, Michigan, August 16-19.

Clancey, W.J. (1990). Implications of the system-model-operator metaphor for knowledge acquisition. In H. Motoda, R. Mizoguchi, J. Boose, & B. Gaines (Eds.) *Knowledge Acquisition for Knowledge Based Systems*. Amsterdam: IOS Press.

Cox, L.A. (1991). Knowledge acquisition for model building. In K.M. Ford & J.M. Bradshaw (Eds.), special knowledge acquisition issue of the *International Journal of Intelligent Systems*, in press.

Diaper, D. (1989). Designing expert systems—from Dan to Beersheba. In D. Diaper (Ed.) *Knowledge Elicitation: Principles, Techniques and Applications*. New York: John Wiley.

Eisenstadt, M., Domingue, J., Rajan, T. & Motta, E. (1990). Visual knowledge engineering. *IEEE Transactions on Software Engineering*, 16(10), October, 1164-1177.

Feigenbaum, E.A. & McCorduck, P. (1983). *The Fifth Generation*, New York: Addison-Wesley

Ford, K.M. (1987). *An Approach to the Automated Acquisition of Production Rules from Repertory Grid Data*. Ph.D. dissertation, Tulane University.

Ford, K., Bradshaw, J.M. & Adams-Webber, J.R. (1991). Knowledge acquisition as a modeling activity. In K. Ford & J. Bradshaw (Eds.), special knowledge acquisition issue of the *International Journal of Intelligent Systems*, in press.

Ford, K.M., Petry, F.E., Adams-Webber, J.R., & Chang, P.J. (1991). An approach to knowledge acquisition based on the structure of personal construct systems. *IEEE Transactions on Knowledge and Data Engineering*, 3, 1-11.

Ford, K.M., Cañas, A.J.., Andrews, E.J., Schad, N., Bezdek, J.C., Brennan, J. & Coffey, J. (1991). NUCES: Nuclear Cardiology Expert System: A preliminary report. *Proceedings of the Fourth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Kauai, Hawaii, June.

Ford, K.M., Stahl, H., Adams-Webber, J.R., Cañas, A.J.., Novak, J. & Jones, J.C. (1991). ICONKAT: An integrated constructivist knowledge acquisition tool. *Knowledge Acquisition Journal*, 3(2), 215-236.

Gaines, B.R. & Boose, J.H. (1991). Standards requirements, sources, and feasibility in knowledge acquisition.*Working notes of the AAAI Workshop on Standards in Expert Systems*. Anaheim, CA: July 14.

Gaines, B.R. & Shaw, M.L.G. (1986a). Interactive elicitation of knowledge from experts. *Future Computing Systems*, 1(2).

Gaines, B.R. & Shaw, M.L.G. (1986b). Induction of inference rules for expert systems. *Fuzzy Sets and Systems*, 8, 315-328.

Goldberg, A. (1990). Information models, views, and controllers. *Dr. Dobb's Journal*, July, 1-4.

Gruber, T.R. (1989). *The Acquisition of Strategic Knowledge*. New York: Academic Press.

Gruber, T.R. (1990). Justification-based knowledge acquisition. In H. Motoda, R. Mizoguchi, J. Boose, & B. Gaines (Eds.) *Knowledge Acquisition for Knowledge Based Systems*. Amsterdam: IOS Press.

Gruber, T.R. (1991). The role of common ontology in achieving sharable, reusable knowledge bases. Stanford Knowledge Systems Laboratory Report No. KSL 91-10, February. To appear in J.A. Allen, R. Fikes, and E. Sandewall (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Second International Conference*. San Mateo, CA: Morgan Kaufmann.

Hix, D. (1990). Generations of user-interface management systems. *IEEE Software*, September, 77-87.

Johnson, N.E. (1989). Mediating representations in knowledge elicitation. In D. Diaper (Ed.) *Knowledge Elicitation: Principles, Techniques and Applications*. New York: John Wiley.

Jones, J.C. (1981). *Design Methods*. New York: John Wiley & Sons.

Kaplan, A. (1963). *The Conduct of Inquiry*. New York: Harper and Row.

Kapor, M. (1991). A software design manifesto. *Dr. Dobbs Journal*, January, 62-67.

Kelly, G.A. (1955). *The Psychology of Personal Constructs*. New York: Norton.

Krasner, G.E. & Pope, S.T. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, August-September, 26-49.

Laland, A., Novotny, R., Enzer, S. & Bortz, J. (1991). *The TIGRE Programming Environment*. Santa Cruz, CA: TIGRE Object Systems.

Larkin, J.H. & Simon, H.A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11, 65-99.

Lenat, D.B. & Guha, R.V. (1990). *Building Large Knowledge-based Systems*. Reading, MA: Addison-Wesley.

Linster, M. & Gaines, B.R. (1990). Supporting acquisition and performance in a hypermedia environment. Presentation at *Terminology and Knowledge Engineering Workshop*, Oct.

McNamee, P. & Celona, J. (1987). *Decision Analysis for the Professional—With Supertree.* Redwood City, CA: Scientific Press.

Moore, E.A. & Agogino, A.M. (1987). INFORM: An architecture for expert-directed knowledge acquisition. *International Journal of Man-Machine Studies*, 26, 213-230.

Musen, M.A. (1988). Generation of model-based knowledge acquisition tools for clinical-trial advice systems. Doctoral dissertation, Computer Science Department, Stanford University. Available as KSL-88-06 and STAN-CS-88-1194.

Norman, D.A. (1988). *The Psychology of Everyday Things*. New York: Basic Books.

Norman, D.A. (1991). Cognitive artifacts. In J.M. Carroll (Ed.), *Designing Interaction: Psychology at the Human-Computer Interface,* New York: Cambridge University Press.

Novak, J.D. & Gowin, D.B. (1984). *Learning How to Learn.* New York: Cambridge University Press.

Parnas, D. (1991). The use of formal methods for computer system documentation. Quoted in *Software Maintenance News,* 9(5), May, 29.

Pearl, J., Geiger, P. & Verma, T. (1988). The logic of influence diagrams. *Proceedings of the Conference on Influence Diagrams for Decision Analysis, Inference, and Prediction.* May 9-11, 1988, Berkeley California, University of California at Berkeley.

Pearl, J. & Verma, T. (1987). The logic of representing dependencies by directed graphs. *Proceedings of the National Conference on Artificial Intelligence,* Seattle.

Puerta, A., Egar, J., Tu, S. & Musen, M. (1991). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. Stanford Knowledge Systems Laboratory Report KSL-91-24.

Rich, E. (1983) *Artificial Intelligence*. New York: McGraw-Hill

Rich, C. & Waters, R.C. (1987). Artificial intelligence and software engineering. In W. E. L. Grimson and R.S. Patil (Eds.) *AI in the 1980s and Beyond: An MIT Survey*. Cambridge, MA: The MIT Press.

Rothenberg, J. (1989). The nature of modeling. In L.E. Widman, K.A. Loparo & N. R. Nielsen (Eds.) *Artificial Intelligence, Simulation, and Modeling*. New York: John Wiley.

Scott, A.C., Clancey, W.J., Davis, R. & Shortliffe, E.H. (1984). "Methods for generating explanations," In B.G. Buchanan and E.H. Shortliffe (Eds.), *Rule-Based Expert Systems*. Reading, MA: Addison-Wesley, pp. 338-362.

Schreiber, G., Akkermans, H. & Wielinga, B. (1990). On problems with the knowledge level perspective. *Proceedings of the Fifth Banff Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Banff, Canada, November.

Schuler, D., Russo, P., Boose, J., & Bradshaw, J. (1990). Using personal construct techniques for collaborative evaluation. *International Journal of Man-Machine Studies,* 33, 521-536.

Shaw, M.L.G. & Woodward, J.B. (1989). Mental models in the knowledge acquisition process. *Proceedings of the Fourth Knowledge Acquisition Workshop for Knowledge Based Systems.* (KAW-89). Banff, Canada, Oct.

Shaw, M.L.G., Woodward, J.B. & Gaines, B.R. (1990). A cognitive framework for knowledge acquisition methodologies and tools. Department of Computer Science, University of Calgary, Internal Report. Calgary, Alberta, Canada.

Shema, D.B. & Boose, J.H. (1988). Refining problem-solving knowledge in repertory grids using a consultation mechanism. *International Journal of Man-Machine Studies*, 29, 447-460.

Shema, D.B., Bradshaw, J.M., Covington, S.P. & Boose, J.H. (1990). Design knowledge capture and alternative generation using possibility tables in *Canard. Knowledge Acquisition Journal*, 2, 345-363.

Sivard, C., Zweben, M., Cannon, D., Lakin, F. & Leifer, L. (1989). Conservation of design knowledge. *Proceedings of the 27th Aerospace Sciences Meeting of the American Institute of Aeronautics and Astronautics*, Reno, Nevada, January 9-12.

Skuce, D. (1991a). A review of 'Building large knowledge based systems' by D. Lenat and R. Guha. *Artificial Intelligence*, in press.

Skuce, D. (1991b). A frame-like knowledge acquisition integrating abstract data types and logic. In J. Sowa (Ed.), *Principles of Semantic Networks,*. San Mateo, CA: Morgan Kaufmann.

Skuce, D. (1991c). A wide spectrum knowledge management system. *Knowledge Acquisition Journal,* in press.

Sowa, J.F. (1987). There's more to logic than predicate calculus. In J. Carbonell and K. Fuchi (Eds.) *Proceedings of the U.S.-Japan AI Symposium,* Tokyo, Japan, December.

Sowa, J.F. (1991). Toward the expressive power of natural language. In J. Sowa (Ed.), *Principles of Semantic Networks,*. San Mateo, CA: Morgan Kaufmann.

Thimbleby, H. (1990). *User Interface Design.* Reading, Mass.: Addison-Wesley.

Tortora, G. (1990). Structure and interpretation of visual languages. In Shi-Kuo Chang (Ed.), *Visual Languages and Visual Programming*. New York: Plenum Press.

van Griethusen, J.J. & King, M.H. (Eds.) (1985). Assessment guidelines for conceptual schema language proposals (ISO TC97/SC21/WG5-3), August.

Webster, D.E. (1988). Mapping the design information representation terrain. *Computer*, Dec., 8-23.

Wielinga, B.J., Akkermans, H., Schreiber, A.TH. & Balder, J.R. (1989). A knowledge acquisition perspective on knowledge-level models. In J.H. Boose & B.R. Gaines (Eds.), *Proceedings of the 4th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop.* Canada, Nov.

Winograd, T. & Flores, F. (1986). *Understanding Computers and Cognition.* Norwood, N.J.: Ablex.

Winston, P. H. *Artificial Intelligence.* 2nd Edition. Reading, Mass.: Addison-Wesley, 1984.

Zachman, J.A. (1987). A framework for information systems architecture. *IBM Systems Journal*, 26(3), 276-292.

Zwicky. F. (1969). *Discovery, Invention, Research through the Morphological Approach.* New York: Macmillan.

Case-Based Knowledge Acquisition in the Preformal Phase

Sonja Branskat
Knowledge Science Institute
Department of Computer Science
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada T2N 1N4

Abstract: This paper presents a system to support a knowledge engineer in formalising case-based knowledge based on a domain independent model for cases. According to the model, each case consists of five components: context, problem, solution trace, solution and evaluation of the solution. The system UFA offers a hypermedial environment in which a knowledge engineer builds a sequence of cases with increasing formal descriptions. It provides functions tailored to direct the formalisation process of the elicitated knowledge. UFA documents the formalisation process and reduces the *media break* in the knowledge acquisition process as far as possible.

1. INTRODUCTION

Many techniques used for knowledge elicitation are based on interviews (Neale, 1988). The expert describes how he works on a (typical) case. His actions, verbal descriptions of what he does and his arguments are protocoled by tape or video records. These protocols form a first knowledge base. They are the closest and detailed knowledge representation of the performed expertise that we have. I refer to this knowledge base as the informal one as the knowledge engineer has not worked on it or changed it. The knowledge engineer evaluates the records by first making a transcript. In transcribing the protocols, a lot of information is lost, e.g. the intonation, gestures etc. The transcripts form a second knowledge base comprising the same (somewhat reduced) knowledge. I refer to this knowledge base as a *preformal* one. Formalizing the knowledge representation step by step the knowledge engineer builds further preformal knowledge bases: He distinguishes between problem solving relevant and irrelevant knowledge, he extracts concepts and their relationships and he visualises information using diagrams and charts. All knowledge bases comprise the same knowledge, but they differ in their formalisation degree. At the end of the formalisation process the knowledge engineer forms a formal representation, that is operational by a computer. Regarding the knowledge representation media, there are two impor-

tant observations: 1) the media are different (tape, paper and disk) and 2) the media are not connected to each other. That means, although the different knowledge bases are semantically strongly related to each other by representing the same knowledge, there is no connection between them that allows to refer from one representation to a more formal or to a less formal representation. This phenomenon is called the **media break** (Karbach and Linster, 1990). The impacts of the media break are serious. We can distinguish between the impacts on the use of the expert system and the impact on updating the knowledge base.

a) Impacts on the use of the expert system:

- To reach the desired information a user has to ask the right questions. That presumes that the user associates the same meaning with terms as the expert who was involved building the expert system. But people (experts, too) use different terms for the same concept or use the same term for different concepts (Shaw, 1988). Because of the ambiguousness it is likely that a user asks a misleading question or that he misunderstands an answer provided by the expert system. A clever user may have a feeling of misunderstanding, but how can he resolve it? Inspecting the knowledge base will not be very helpful, if a user is not used to reading rules, constraints network, frames etc. More helpful would be an inspection of the inherent informal or preformal knowledge. But no pointer exists from the formal to the informal representation or to any of the preformal representations, worse no preformal or informal knowledge base has been delivered with the system.
- The access to the preformal knowledge base (including the informal) is missed as well, if a user does not understand a term used by the expert system. Terms that describe noises, colours or pictures are not very precise. Having access to the preformal knowledge bases a user could listen to a tape or see a picture.
- Due to the missing of preformal knowledge bases a user cannot refer to the context from which a term or statement was derived, e.g. the case. A medical expert system, for example, may propose a very rare tropical disease. A puzzled doctor practicing in North Canada could restore his confidence in the expert system, if he had access to the case the expert system refers to: A case where a person with similar symptoms has been for a business trip to the tropics.

b) Impact on updating the knowledge base:

-An expert system needs to be updated. Any improvement or expansion of an expert system has to be made according to the existing knowledge base. To reach a consistent expansion documentation of the knowledge formalisation process is necessary or at least very helpful. But as we saw neither the informal representation nor any preformal

representation is available due to the media break and the loss of every representation except the formal one.

In the recent years hypermedia systems have offered the integration of different knowledge representation including informal representation such as video tapes and pictures into a hybrid knowledge base. One hypermedia system that is used in knowledge acquisition is HYPER-KSE (Gaines and Linster, 1990; Linster and Gaines, 1990). It reduces the impact of the media break on the use of an expert system. Here the idea of using a hybrid knowledge base is extended to the whole formalisation process of case-based knowledge.

2. CASES

Cases as a knowledge source have the following advantages:

a) A case is a very familiar concept to an expert. He is used to talk about cases, and he uses his experience gained from previous cases to work on new cases (Riesbeck, 1989).

b) There are several elicitation techniques for case-based knowledge (Neale, 1988).

c) A case provides the context in which the problem solving knowledge is applied (Becker, 1988). The context of knowledge is an important point, because there is strong evidence that knowledge is context dependent (Becker, 1988; Winograd and Flores, 1989; Compton and Jansen, 1990; Shaw and Woodward, 1990).

d) Case-based knowledge has a broad scope. It is used in *conventional* expert system as well as in Case-Based Reasoning (CBR) systems. By conventional, I mean expert systems that operate with rules, frames or constraints etc. They use case-based knowledge to extract concepts, relationships between concepts or generalize cases to rules but they do not reason with cases, whereas CBR systems use cases to solve new cases (Kolodner, 1988). The formalisation of case-based knowledge here does not rely on either kind of system.

2.1 Formalizing Cases

Formalization as it is defined in (Duden, 1983; Brockhaus-Enzyklopädie, 1989) applied to cases means:

1) to describe the case verbally,

2) to structure case descriptions,

3) to standardize the case descriptions, that means to reach a systematic appearance of cases descriptions and

4) to resolve misunderstandings due to ambiguous formulations.

A first step towards structuring is to provide a case model. A domain independent case model provides at least the same initial structure on each case. Additionally it systematizes the appearance of the case representations. Further structuring and systematic appearance is reached by providing the knowledge engineer with a fixed set of methods to describe a case. The discovery and resolving of ambiguities is a difficult task. UFA only provides functions that help the knowledge engineer to discover different uses of a same word and the use of different words for the same concept.

2.2 Case Model

In the literature there is no agreement on what a case exactly means: "The word case just refers to an experience." (Riesbeck, 1989, p. 11). Influenced by CBR systems (Kolodner, 1988) the proposed case model here structures each case into five components: **context, problem, solution path, solution** and **evaluation of the solution**. Cases from the two domains *Office Planning* and *Business Graphs* show that this case model provides a complete description of cases and vice versa that the case components contribute essentially to a case description (Branskat, 1990). The example given in 3.2 illustrates the case model and the meaning of the components.

3. THE SYSTEM: UFA

UFA is implemented in HyperCard and runs on a Macintosh. Actually the system restricts the knowledge engineer to pictures, graphics, and free text, i.e. it does not support video or tape access in this version. The system supports a structured attribute/value representation of cases, which is a frequently used representation of cases in CBR systems (Kolodner, 1988).

3.1 Design of UFA

In UFA each case is represented as a stack* of at least 6 cards: one title card and one card for each case component according to the case model (see Figure 1). Annotation cards for extending description as well as for pictures are added to the stack on request. The case descriptions, informal, preformal and formal, build a formalisation sequence (see Figure 1) that connects different case descriptions of one case to each other.

---

* HyperCard (HyperCard, 1988) is like an electronic index card system. The information is basically stored in the cards. Cards are sorted in stacks. Pointers connect card entries.

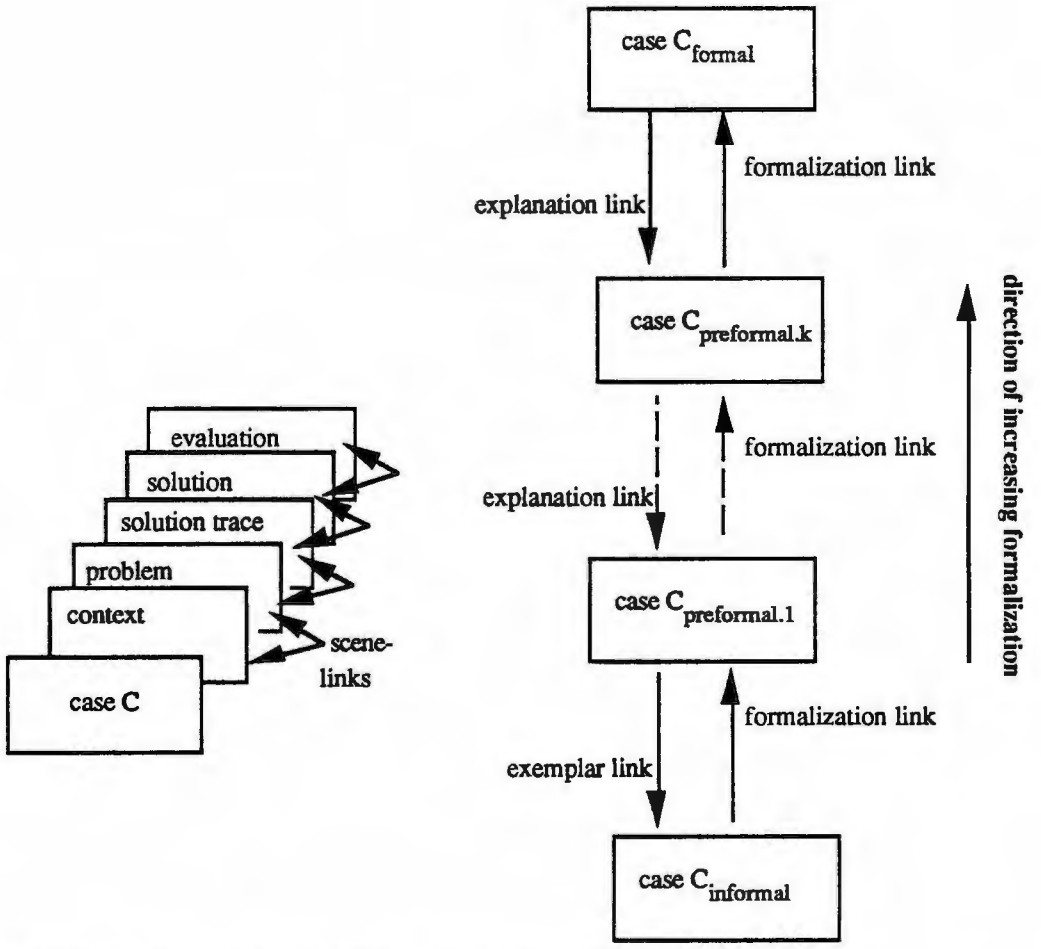


Figure 1: Case model and formalization sequence.

In the formalization sequence successive descriptions of the same case *C: case* $C_{informal}$, *case* $C_{preformal.1}$, ..., *case* $C_{preformal.k}$, ... , *case* C_{formal} (see Figure 1) have an increasing degree of formalization. The sequence is part of the documentation of the formalization process.

The used data structure is based on the Memory Organisation Packages (Schank, 1985) used in several CBR systems to represent cases (Kolodner, 1988). Scene-links connect the components of a case. Formalization links point from a case description (e.g. case $C_{preformal.k}$) to the next more formal case description (e.g. case $C_{preformal.k+1}$). Explanation links point to a case description (e.g. case $C_{preformal.k}$) with the next less formal description (e.g. case $C_{preformal.k-1}$). The link is called *explanation link* as in general the less formal description is richer and more comprehensive. It can be refereed to as an explanation of a more formal description of the same knowledge. For example the acoustical demonstration of a noise is much more precise than its verbal description. Exemplar links connect the case description of the first formalization degree (case

$C_{preformal.1}$) with the initial informal description (case $C_{informal}$). This link indicates the end of the explanation sequence.

3.2 Illustrative Example

The following example belongs to the domain of business graphics. Business graphics visualize quantitative relations between data with graphical techniques to obtain a sound information presentation. A very common used graph type are circle graphs, bar (horizontal and vertical) and curve graphs.

3.2.1 Case Model Applied on Cases taken from the domain of Business Graphs

The **context** describes among others where the chart is published, e.g. in a periodical or in a presentation; how large the graph is, constraints e.g. on the choose of colours, etc.

The **problem** is the appropriate choice of a graph type to stress a given statement and a data set regarding the context.

The **solution path** describes how an expert worked out a solution. How did he chose a chart type: did he exclude types, had he refereed to a previous case etc.

The **solution** comprises the graph, its heading, the choice of colours or patterns, its labeling etc.

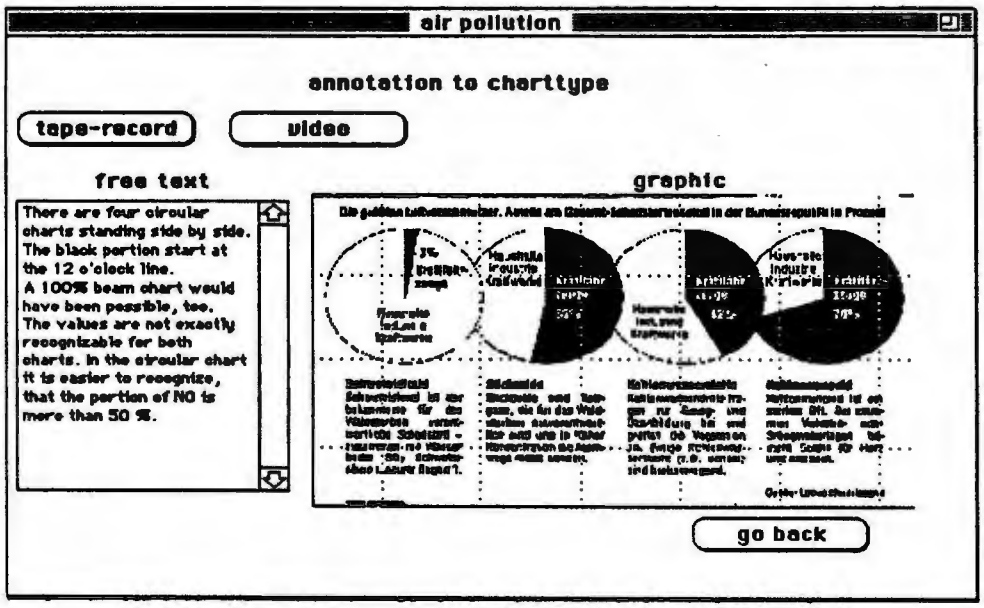


Figure 2: The first formalization of the case air pollution*: a sub-component of the component solution. The knowledge engineer used free text and a scanned graphic for the description.*

The **evaluation of a solution** argues why the solution is the best one or the best one in the given context or why it is a bad solution. Bad solutions are likely to be generated in

CBR systems in order to avoid repetitions of bad solution some CBR systems evaluate the generated solutions (Hammond, 1989).

3.2.2 Representation in UFA

At the outset a knowledge engineer identifies the components: context, problem, solution path, solution and evaluation and generates a new MOP, e.g. generates a new stack, in UFA (case $C_{informal}$) to describe the components. He describes the component *solution* with the scanned graphic as an informal description (see Figure 2).

In a second step the knowledge engineer generates a new MOP (case $C_{preformal.1}$) and connects it with a formalization link to the initial case description (case $A_{informal}$). On this first preformal layer he structures the domain independent component *solution* into domain dependent structural sub-components: the chart type that is used, the colours that are used and the labelling of the chart. Using the function *add sub-component* (see Figure 3) he generates new fields to describe these sub-components. Using the function *add annotation* (see Figure 3) he creates an annotation card for the selected sub-component of the component *solution* and describes it with a picture and free text (see Figure 2 for the sub-component *chart type*). With the function *go back* (see Figure 2) UFA returns from the annotation card to the component *solution*. Buttons labelled *A* (see Figure 3) indicate, that there is an annotation card for this sub-component.

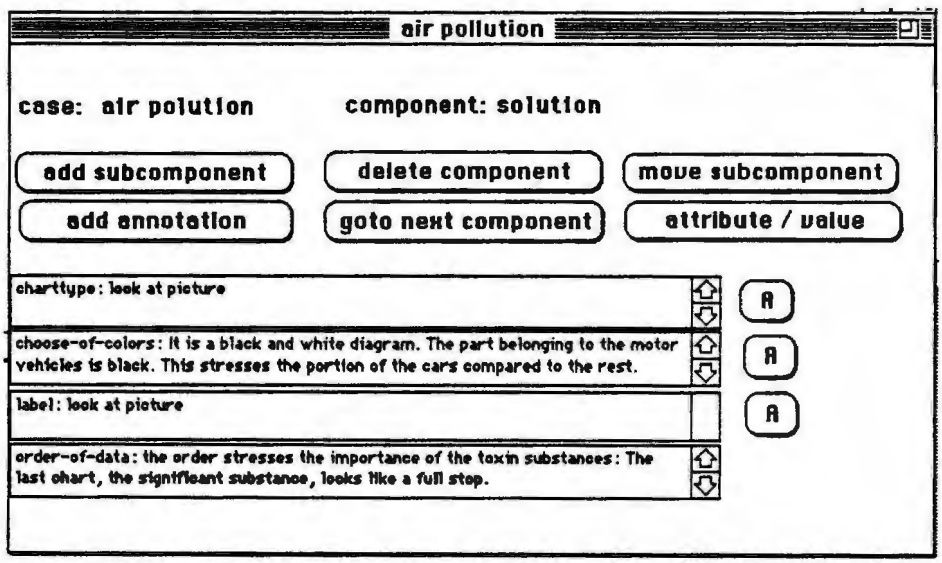


Figure 3: The component solution of the case air pollution*: the formalization links and the textfields for the already generated sub-components.*

At the outset of the third step the knowledge engineer creates a new MOP (case $C_{preformal.2}$) and connects it with an explanation link to its more informal version (case $C_{preformal.1}$). He uses a formalization link to connect (case $C_{preformal.1}$) to its more formal version (case $C_{preformal.2}$). He decides to copy the sub-components of the old description (case $C_{preformal.1}$) to the components of the more formalized case description (case $C_{preformal.2}$) and to edit them in order to save typing. The knowledge engineer creates a new sub-component of the component *solution* to describe the solution in a more formal way by adding the order of the data that are presented. He describes the new structural component *order-of-data* directly on the solution card using the text field popping up with the creation of a new sub-component.

A few formalization steps later the knowledge engineer describes the sub-component *chart type* with *circle-chart* and decides, that *multiple circle-chart* should be a value of the attribute *chart type*. He declares this in UFA using the function *attribute/value* (see Figure 3) and expanding the range of values of this attribute. The case is formalized if all components are described with attributes and values (see Figure 4). The formalization is complete, as now all the relevant knowledge enclosed in graphics and text has been transformed into a formal, computer accessible and retrievable notation.

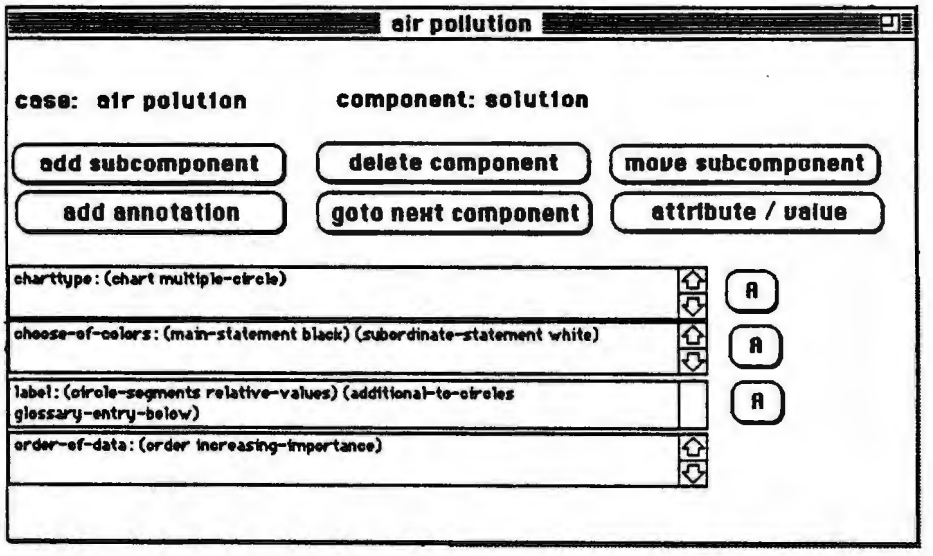


Figure 4: The component solution of the case air pollution*: a complete formalized version.*

3.2.3 Well-directed Support for Formalisation in UFA

Formalisation points at: structuring of case descriptions, standardizations and systematic appearance of descriptions and resolving ambiguities.

structuring: The case model provides a first common structuring in the five case components: context, problem, solution path, solution and evaluation. Each further structuring is in terms of sub-components.

standardizations and systematic appearance: A consistent structure of the case descriptions contributes to a standardization, too. To reach a further standardization the knowledge engineer determines one order of the sub-components for all the descriptions of a case and the system rewrites the selected order to all case descriptions. A further step versus standardisation is achieved by the assigned format of the formal description as a structured list of attribute/values pairs. The knowledge engineer declares sub-components as attributes and specifies their values. The systems provides functions to check if the values of the attributes in the case descriptions are valid.

resolving ambiguities: While formalisation takes a lot of time the knowledge engineer may use different terms for the same meaning in various case descriptions of a same case or in different cases or he can have used one term in different meanings. He should reassure himself that the case descriptions (at least the final, the formal description) are free of ambiguities. For selected cases and case components the systems presents an overview over the sub-components. The knowledge engineer checks their description in order to see if the names are used in a consistent manner. If he discovers different terms meaning the same he declares them as synonym and select one of them to substitute the others. The system protocols the synonyms. If the knowledge engineer finds a term with different meanings he renames it.

4. CONCLUSIONS

The aim of this work was to design and implement a support system for the formalisation of case-based knowledge that documents the transformation of the knowledge representation. At the outset I developed a case model and a data structure to represent different descriptions of a case based on the case model. The system functions are specifically tailored to support the formalisation process. The system is implemented and was tested in the domain of business graphs. The final formal representation should be useful as a starting point for CBR systems as well as for conventional expert systems or for initiating further knowledge elicitation with e.g. a tool like KSSO (Shaw and Gaines 87).

UFA is a prototype and this study has shown that it is promising to extend and to integrate film sequences and tape records into such a system. A second extension is the integration of a system like UFA in an expert system and the support of further formal representations.

5. ACKNOWLEDGEMENTS

I am grateful to Marc Linster, Barbara Becker, Thomas Christaller and all the other members of the Artificial Intelligence Research Division of the German National Research Centre for Computer Science (GMD) and Brian Gaines for making valuable comments on this or a previous version of this article.

6. REFERENCES

Becker, Barbara (1988). Towards a concept for case-based knowledge-acquisition; In *Proceedings of the European Knowledge Acquisition Workshop (EKAW'88).*

Branskat, Sonja (1990) *Fallbasierte Wissensakquisition: Akquisition und Repräsentation von Fällen in der präformalen Phase der Wissensakquisition* Diplomarbeit, FernUniversität-Hagen.

Brockhaus-Enzyklopädie 19.te Auflage (1989); Mannheim: Brockhaus.

Compton, P. and Jansen, R. (1990) A philosophical basis for knowledge acquisition; *Knowledge Acquisition 2* , 241-257.

Duden: Deutsches Universal Wörterbuch; (1983) Bibliographisches Institut Mannheim, Wien, Zürich.

Gaines, Brain and Linster, Marc (1990) *Development of Second Generation Knowledge Acquisition Systems;* Introduced to EKAW90.

Hammond, Kristian and Converse, Timothy (1989) Panel discussion on case representation; In *Proceedings of a Workshop on Case-Based Reasoning,* Morgan Kaufmann Publisher, Inc.

HyperCard (1988) *HyperCard Script Language Guide The HyperTalk Language;* Addison-Wesley (July 1988)

Kolodner Eds. (1988) *Proceedings of a Workshop on Case-Based Reasoning*

Linster, Marc and Gaines, Brian (1990)*Supporting Acquisition and Interpretation of Knowledge in a Hypermedia Environment,* Tech. Rept. 455, Arbeitspapiere der GMD, GMD, July.

Neale, I.M. (1988) First generation Expert Systems: a review of Knowledge Acquisition Methodologies; *The Knowledge engineer review 3* , pp. 105-146.

Riesbeck Ch. 1989 Inside *Case-Based Reasoning;* Lawrence Erlbaum Associates, Hillsdale, New Jersey (1989).

Schank, Roger C. (1985) *Dynamic memory;* Cambridge University Press.

Shaw, Mildred L.G. and Gaines, Brian (1987) An Interactive Knowledge Elicitation Technique Using Personal Construct Technology; In *Knowledge Acquisition for Expert Systems*; Plenum Press; ed. Kidd, A.; New York.

Shaw, Mildred L.G. (1988) Problems of Validation in a Knowledge Acquisition System using Multiple Experts; In *Proceedings of EKAW88,* ed. Boose, John H., Gaines, Brian R., and Linster, Marc, GMD, St. Augustin, pp. 5/1 - 5/15.

Shaw, Mildred L. and Woodward, Brian (1990) Modeling expert knowledge; *Knowledge Acquisition 2,* pp. 179-206.

Winograd, Terry and Flores, Fernando (1986) *Understanding Computers and Cognition;* Ablex Publishing, New Jersey.

Ripple down rules: possibilities and limitations

P.Compton, G.Edwards*, B.Kang, L.Lazarus*, R. Malor*, T.Menzies, P.Preston.
A.Srinivasan, S.Sammut.

School of Computer Science and Engineering, University of New South Wales,
PO Box 1, Kensington NSW, Australia 2033,
email compton@spectrum.cs.unsw.oz.au
and
Department of Chemical Pathology, St Vincent's Hospital
Darlinghurst NSW, Australia 2010

ABSTRACT

A major problem with building expert systems is that experts always communicate knowledge in a specific context. A knowledge acquisition methodology has been developed which restricts the use of knowledge to the context in which it was provided. This method, "ripple down rules" allows for extremely rapid and simple knowledge acquisition where the time required to incorporate each new piece of knowledge remains constant regardless of the size of the knowledge base. An expert system based on this approach and built by experts without the help of a knowledge engineer is now in routine use. This paper reviews what has been achieved using the approach, its problems and potential.

A PHILOSOPHY OF KNOWLEDGE

Ripple down rules are based on a specific philosophical view of the nature of knowledge (Compton and Jansen, 1990). We make no apology for this, for we would suggest that approaches to building knowledge based systems that eschew any philosophical consideration are in fact strongly dependent on and limited by such assumptions. The prevailing assumptions are implicitly Platonic, in that it is assumed knowledge has some concrete reality, it is some sort of stuff that can be "extracted" or "mined" from experts' heads. If you can't get enough of this knowledge stuff to make the expert system work properly it is because the expert can't properly report on what is in his head. The problems of building knowledge bases would disappear if only acquisition could acquire deep (true) expert knowledge. Clear cut evidence that this is not the right way to approach knowledge is provided by Shaw(Shaw 1988) who has demonstrated that experts may have quite different and apparently inconsistent knowledge about a domain but are able to freely communicate with each other. Observation of experts during the maintenance phase of a expert system development suggests that experts never provide information on how they reach a specific judgment. Rather the expert provides a justification that their judgement is correct. The justification they provide varies with the context in which they are asked to provide it (Compton, Horn et al., 1989; Compton and Jansen, 1990a; Compton and Jansen, 1990b). The context will vary with the questioner. An expert will justify their expert judgement in quite a different ways when queried by fellow experts, trainees, knowledge engineers, lay people. The context depends on the framework in

which questions are asked. If the knowledge engineer believes that "real" knowledge is causal the expert will provide justifications in terms of causality. If the knowledge engineer favours heuristics, the expert will provide heuristics. It does not seem to be correct to approach knowledge acquisition as the task of getting to the deep and more inaccessible knowledge. Rather experts always justify their judgement with equal ease. The problem is that this justification varies with the context and has to be "engineered" to fit in with the other knowledge in the knowledge base.

The best way of understanding knowledge seems to be to view knowledge and knowledge bases as models (Clancey, 1989). As Clancey points out the basic feature of a model is that it can simulate reality, behave like reality according to the expectations of the users. From this point of view a knowledge base is not a model of the expert's knowledge, both the knowledge base and the expert's knowledge are models of the domain. This is essentially traditional philosophy. Knowledge is different from reality or the thing in itself, knowledge is the way in which the knower relates to or "understands" the known. The concept of model captures this same dichotomy. The model is not the thing but it behaves like the thing, not in a absolute sense but according to the expectations of the model or knowledge users. Clancey points out that even a set of rules which merely capture heuristics about the domain, in some sense model the domain and behave like the domain. There are of course many different kinds of models which one can build, and may different criteria as to what is an acceptable simulation of reality.

From the modelling point of view the expert's "knowledge" or justification in context is the creation of a model to explain how reality works to fit with or replace other models of reality within the constraints of certain modelling paradigms. This modelling view is in turn consistent with Popper's falsification approach to the development of knowledge (Popper, 1963). Knowledge always consists of hypotheses which can never be proven correct, at most they can be proven incorrect and be replaced by other "less false" hypotheses. The hypothesis is a model and as such is intrinsically different from reality. The model is also constructed within a particular paradigm, particular context and so will have to changed or replaced to cope with other points of view. None of this implies a relativist position. The "truth" of knowledge is found in Lonergan's concept of "insight" (Lonergan, 1959). Essentially "knowledge" is the model we make of reality to express our insight into reality, our experience of some intelligibility in reality. Conversely "insight" is the process or act of recognising that the knowledge we have constructed is a model of reality, that it makes sense of reality in some way.

Clancey (Clancey, 1989) and Gaines (Gaines, 1991) provide excellent summaries of these ways of thinking about knowledge from the perspective of AI practitioners rather than the opposition to AI perspective (Dreyfus and Dreyfus, 1988). They touch on issues beyond the scope of this paper, suggesting that knowledge does not reside in the head but in its expression in some medium, and that knowledge has an important social dimension. The important aspect of these different approaches to knowledge is not that they question the possibility of artificial intelligence but that they open up different approaches to acquiring, representing and using knowledge. Ripple down rules which will be outlined in more detail below is one attempt to handle knowledge acquisition and use from this perspective. Repertory grid methods such as KSSO can also be viewed as capturing similar philosophical concerns (Gaines and Shaw, 1990).

An implication of the argument above is that any attempt to understand how an expert reaches decisions by asking questions or interviewing in essence asks the expert to provide a justification for his decisions within the context of the interviewing or questioning paradigm. The model the expert helps the knowledge engineer construct is always influenced by the framework the knowledge engineer is working in, the tools available etc. Since this always happens it can be argued that there may be advantage in having fairly strong tools, which allow the expert very limited but simple choices in providing knowledge, but which are adequate to

model the domain. The strong tool may have considerable advantage over the weak methods which leave it to the expert and knowledge engineer to structure the knowledge. With the weak methods the structure of the knowledge is implicit and therefore difficult to maintain and little advantage has been gained in trying to model the experts knowledge because the expert has been in fact justifying his or her judgement within the framework of the knowledge acquisition tools and techniques and the knowledge engineer's concerns. (Alain Rappaport, personal communication).

We have developed a knowledge acquisition methodology "ripple down rules" which attempts to recognise that the knowledge the experts provide is only a justification in context and that this justification will be most reliable if used only in the same context (Compton and Jansen, 1990a ; Compton and Jansen, 1990b). The method described below can be considered independently of the analysis above and it stands or falls on its performance. However, this section started with the strong claim that philosophical assumptions influence the development of practical methodologies. The ripple down rules method was explicitly based on philosophical assumptions which were noted in 1987 before the method was developed (Compton 1989). The philosophical analysis above, although now expanded, is not post hoc.

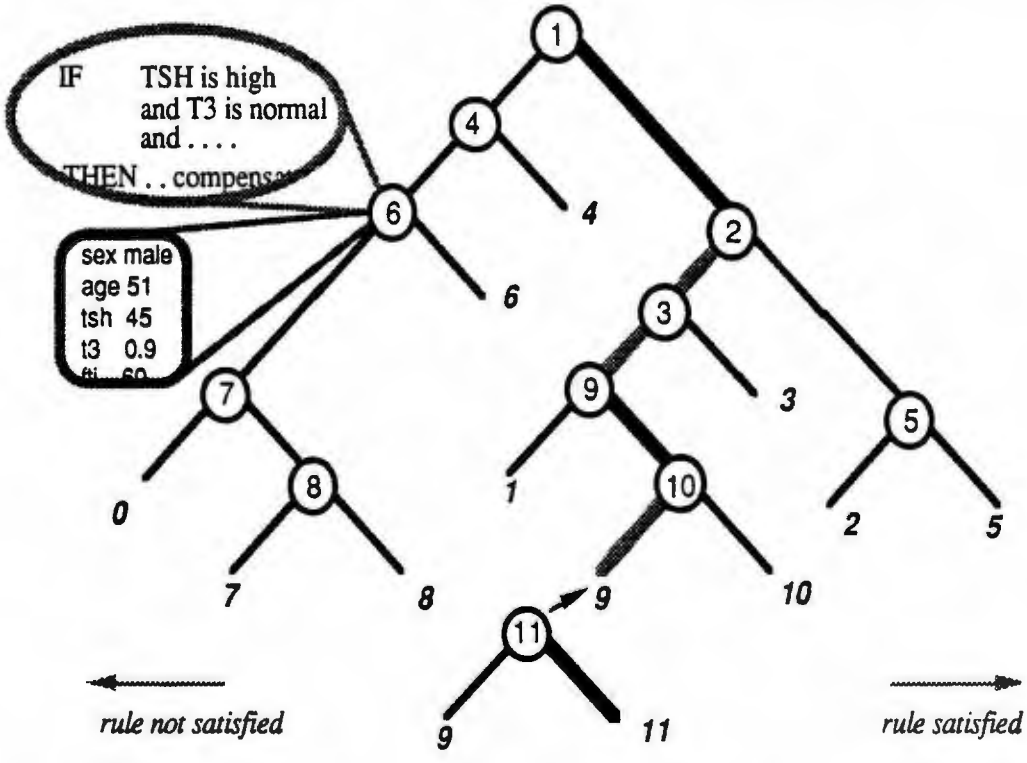


Fig 1 - A ripple down rule knowledge base. Each node in the tree is a rule with any desired conjunction of conditions. Each node also has a case associated with it, the case that prompted the inclusion of the rule. In this example rules are numbered as are the leaves of the tree to indicate which rule the classification comes from when a case traverses the tree. The classification comes from the last rule that was satisfied by the data. The heavy line illustrates a path through the knowledge base for a case giving interpretation 9. If this is the wrong interpretation and a rule has to added to give interpretation 11, it would be joined onto the unsatisfied branch of rule 10 as shown, as this was the exit point from the knowledge base for that particular case differences

CONTEXT IN RIPPLE DOWN RULES

Most of the experiments we have carried out have been in the domain of providing clinical interpretations for pathology reports. That is, a comment is appended to the report explaining what the results mean, what tests should or should not be ordered in the future etc. The crucial component of the context in this domain is that the expert is justifying why their new interpretation is better that the interpretation given for the case (perhaps no interpretation). If the expert was justifying why their interpretation was better than another person's, they would try provide the justification in terms of what they assumed to be the other person's beliefs and assumptions etc. If one considers the justifications provided to a patient versus when a colleague queries the expert's interpretation of a report, this attempted guessing of the other person's knowledge or reason for the query is obvious. In the case of the expert system, the knowledge behind the interpretation does not have to be guessed, it is all the knowledge in the knowledge base that has determined the interpretation provided by the expert system. This includes both rules that have actually been satisfied by the data, and rules that have been candidates to fire during the inference process but have not been satisfied by the data. In other words we want the new rule that is being added to be applied to a set of data only if the data is able to satisfy the rules that previously provided the wrong interpretation and if the data does not satisfy other rules also not satisfied in providing the earlier wrong interpretation.

RIPPLE DOWN RULE STRUCTURE

To capture context in this way the expert system is built as a tree with a rule at each node with two branches depending on whether the rule is satisfied or not by the data being considered. Any new rule that is added in response to a wrong interpretation, is attached to the branch at which the expert system terminated, thus making a new node. For example, in a one rule expert system, if the rule gives an interpretation which the expert thinks is wrong the new rule will be attached to the satisfied branch of the first rule. On the other hand if the one rule expert system fails to give an interpretation the new rule added will be attached to the unsatisfied branch. The tree then evolves over time as the expert corrects wrong interpretations. Note that each node of the tree is a rule of any desired complexity and that the interpretation produced from the tree is that from the last rule that was satisfied by the data (Fig 1).

This knowledge base structure greatly facilitates rule addition. Normally when rules are added or changed the performance of the expert system has to be checked against cases. One approach to this is to use "cornerstone cases", cases for which the rules were previously changed (Compton, Horn et al. 1989)). One has to check all such cases. Here this is unnecessary. The only cornerstone case that can be misinterpreted by the new rule is the case associated with the rule that last fired, the rule that gave the wrong interpretation, which the new rule is going to correct. Rather than check this case, the expert can select conditions for the new rule from a list of differences between the case for which the rule is added and the cases associated with the last rule that fired giving the wrong interpretation.

For example:

old case	**new case**
TSH high	TSH high
T3 low	T3 low
FTI normal	
	TT4 high

The expert must choose either or both the conditions

FTI NOT normal

TT4 high

as conditions in the rule and can optionally chose any of the common conditions to make the rule intelligible. Such a rule is guaranteed to work on the new case but not the old case, so no further checking is required or relevant. This simple selection of differences is the basis of the claim that this is a strong method. If the expert wishes to interpret a case there must be sufficient reason for this in the way the case differs from another case for which the old interpretation is correct. It is irrelevant how the expert carries out her or his reasoning; however they reason, there must be a difference between the cases on which their reasoning is based. Similarly the structure of the expert system's knowledge can remain hidden, the expert does not need to know about it. As will be discussed later there may be other underlying knowledge structures which can present the expert with cases differences.

It can be noted that this method of differences relates to other knowledge acquisition methodologies based on personal construct theory, in turn based on the assumption that experts are good at identifying differences in specific contexts (Gaines and Shaw 1990). In contrast to these other methods which ask the expert to think of a difference that is important, here the expert has to identify which differences identified by the system are important. The repertory grid methods are concerned with difference discovery, ripple down rules with difference selection.

EXPERIENCE

Our initial experiments were concerned with redeveloping GARVAN-ES1. We found that no knowledge engineering was required and that 'ripple down rules' could be simply added to the knowledge base as provided by the expert. This results in knowledge acquisition at least 40 times as fast as that required for a conventional version of the same knowledge base, with the same knowledge engineer/expert involved.(Compton and Jansen 1990a & b).

We have developed a more general purpose ripple down rule system which has recently been put into routine use in the Department of Chemical Pathology St.Vincent's Hospital, Sydney. It is intended that this system will be able to deal with interpretative reports for all of Chemical Pathology. Chemical Pathology covers hundred of different analytes and individual reports may include time series data on twenty or more analytes. The St.Vincent's laboratory conducts about one million analyses per year and issues 500 reports per day. The St. Vincent's system has two modules, a batch mode version that runs automatically to append comments to reports and a maintenance version which shows the case differences and which the expert uses to add new rules. Both versions are written in C and run on a Vax under VMS. An example case interpreted using the system is shown in Fig 2.

The rule shown in Fig 2 and the rules in Fig 3 indicate a number of features of the system. Firstly, although the data is initially classified in terms of the laboratories normal ranges, this is not always sufficient to discriminate between cases. For example, although the upper limit of normal for the analyte TSH is 6 mU/L, experts will often consider results of up to 10 mU/L or perhaps more as normal. In a normal expert system this has to be handled by borderline regions as in GARVAN-ES1 (Horn, Compton et al. 1985) or by some sort of fuzzy or probabilistic classification. With ripple down rules these strategies are not required and the cut-off values can be redefined in the context. For example in the two cases given above the TSH

05-Jul-91 ST. VINCENT'S HOSPITAL 12:58 PM
DEPARTMENT OF CHEMICAL PATHOLOGY

Patient ..... : LAIDUP,I.M. Male, 72 years

M.R.N. ..... : 02-62-64 Ward : CH ITU

Test	Range	Units	BLOOD 746406 12:55 03 Jul 1991	BLOOD 746573 07:12 04 Jul 1991	BLOOD 746590 08:46 04 Jul 1991	BLOOD 746986 07:14 05 Jul 1991	BLOOD 747004 09:15 05 Jul 1991
Sodium	137-146	mmol/L	....	....	139	....	140
Potassium	3.5-5.0	mmol/L	....	....	4.0	....	3.8
Chloride	95-105	mmol/L	....	....	103	....	104
Bicarbonate	24-31	mmol/L	....	....	23*	....	22*
Urea	3.0-8.5	mmol/L	....	....	8.7*	....	8.6*
Creatinine	0.06-0.12	mmol/L	....	....	0.14*	....	0.13*
Inorg. Phos.	0.70-1.40	mmol/L	....	....	....	....	0.90
Magnesium	0.85-1.05	mmol/L	....	....	....	....	0.86
Calcium	2.10-2.60	mmol/L	....	....	....	....	2.16
Albumin	36-47	g/L	....	....	33*	....	34*
Tot. Protein	66-82	g/L	....	....	62*	....	65*
Bilirubin	< 18	umol/L	....	....	15	....	12
Alk. Phos.	30-100	U/L	....	....	52	....	64
ALT	< 30	U/L	....	....	33*	....	40*
GGT	< 35	U/L	....	....	46*	....	55*
pH	7.35-7.45		7.51*	7.40	....	7.43	....
PCO2	32-45	mmHg	28*	37	....	34	....
PO2	75-105	mmHg	107*	93	....	68*	....
Bicarb.	24-31	mmol/L	22*	23*	....	22*	....
Base Excess	-3-+3		+1	-1	....	-1	....

Results consistent with:
Resolution of respiratory alkalosis.
Evolving hypoxaemia.

Rule trace: *(only satisfied rules shown)*

```
314:     true
CURR(BLOOD_PH) is NORMAL
AND
 CURR(BLOOD_PO2) > 59.00
AND
 CURR(BLOOD_PO2) is LOW
370:    rule false
397:    rule false
485:     true
 MIN(BLOOD_PCO2) < NORMAL
```

Fig 2 illustrates a typical case interpreted by the expert system. The clinical interpretation directly follows the data. The rule trace was provided by the maintenance module, the batch run time system produces only the interpretation. This feature of the maintenance module is not normally used in arriving at a new rule. Two rules fired on this case with the interpretation provided by the last rule that fired, rule 485. The CURR function returns the value for the most recent measurement for that analyte. For the analytes above this is the 5th July 7:14 sample on the . The minimum BLOOD_PCO2 is the first result. Notice that the expert has chosen to specify that BLOOD_PO2 is low but above a certain value.

was high. If in the old case the value was 12 mU/L and the new case 7mU/L, the expert can incorporate into the new rule TSH < X where X is chosen so that 7 <= X < 12. The feature extractor which classifies TSH as high is not changed, but only cases with TSH values below the new cutoff will be interpreted by rules in the new context. Within this context the TSH cut-offs can be further refined. This technique is both more powerful and more simple than attempting to arrive at a universal classification or feature extraction system that will apply to every context.

```
RULE 369
IF ((CURR(AGE)*(-0.27)) + 94) - CURR(BLOOD_PO2) > 0.00 & MIN(BLOOD_BIC) <
NORMAL THEN

RULE 379
IF  CURR(BLOOD_PH) is NORMAL & CURR(BLOOD_PCO2) is LOW & CURR(BLOOD_PO2) is
HIGH & CURR(BLOOD_BIC) is LOW  THEN

RULE 410
IF TDIFF(CURR(BLOOD_TSH),MAX(BLOOD_TSH) < 30.00 THEN

RULE 424
IF AVG(BLOOD_PO2) < 60.0 THEN

RULE 452
IF VAL(BLOOD_RGLU, MIN(BLOOD_ST)) < 5.80 & CURR(BLOOD_DYNT) is "GT100" &
MAX(BLOOD_RLU) < 10.50  THEN
```

Fig 3. Example rules showing some of the built in functions used. Note in RULE 369 a mathematical expression that the expert has included. The CURR function returns the value for the most recent measurement of the analyte while MIN, MAX and AVG return the values implied in their names. The VAL function return the value for the first parameter at the time specified by the second parameter; in rule 452 it returns the value of BLOOD_RLU at the time of the minimum BLOOD_ST. TDIFF returns the time difference between the two time points specified. Notice in RULE 369 that ordinal relationships exist not only between numerical values but between the primary classifications of the data.

The second feature of the rules in Figs 2 and 3 are the functions used to deal with temporal data. These functions allow the expert to make up rules about:

- The most recent value available for an attribute
- The maximum and minimum values of attribute over the time period
- The average value of an attribute over a time period, and the nett change in its values over that period (to obtain rates of change).
- The value of an attribute at the time instant specified by the current/maximum/minimum value of another attribute (or the nearest available one to this time instant).
- The time difference between a range of possible options. For example, that between the nearest available value of an attribute and the time instant specified by the current/maximum/minimum values of another attribute. Alternatively, the time difference between the current/maximum/minimum values of one attribute and the current/maximum/minimum values of another.

The experts can also include in rules complex numerical expressions relating the values returned by the functions.

These functions are all built-in and so a programmer is required if further functions are to be included. The particular functions included here were arrived at after a conventional knowledge acquisition exercise of discussing with experts what features they need to identify in the data. With ripple down rules in their current development this remains the major task for a knowledge engineer, identifying and programming such functions. As will be discussed later this is a limitation of ripple down rules. However it should be noted that this is still a major advance on conventional expert systems. Feature extraction only appears as a problem here because knowledge engineering has become so simple. In a conventional system it is extremely difficult to deal with temporal data like that shown in Fig 2 (Coiera 1989). The system here has been able to deal with a wide range of temporal data using these very simple functions because the expert is able to redefine critical values for these functions in context. It should be noted that this system is already in routine use and the expert is not assisted by a knowledge engineer.

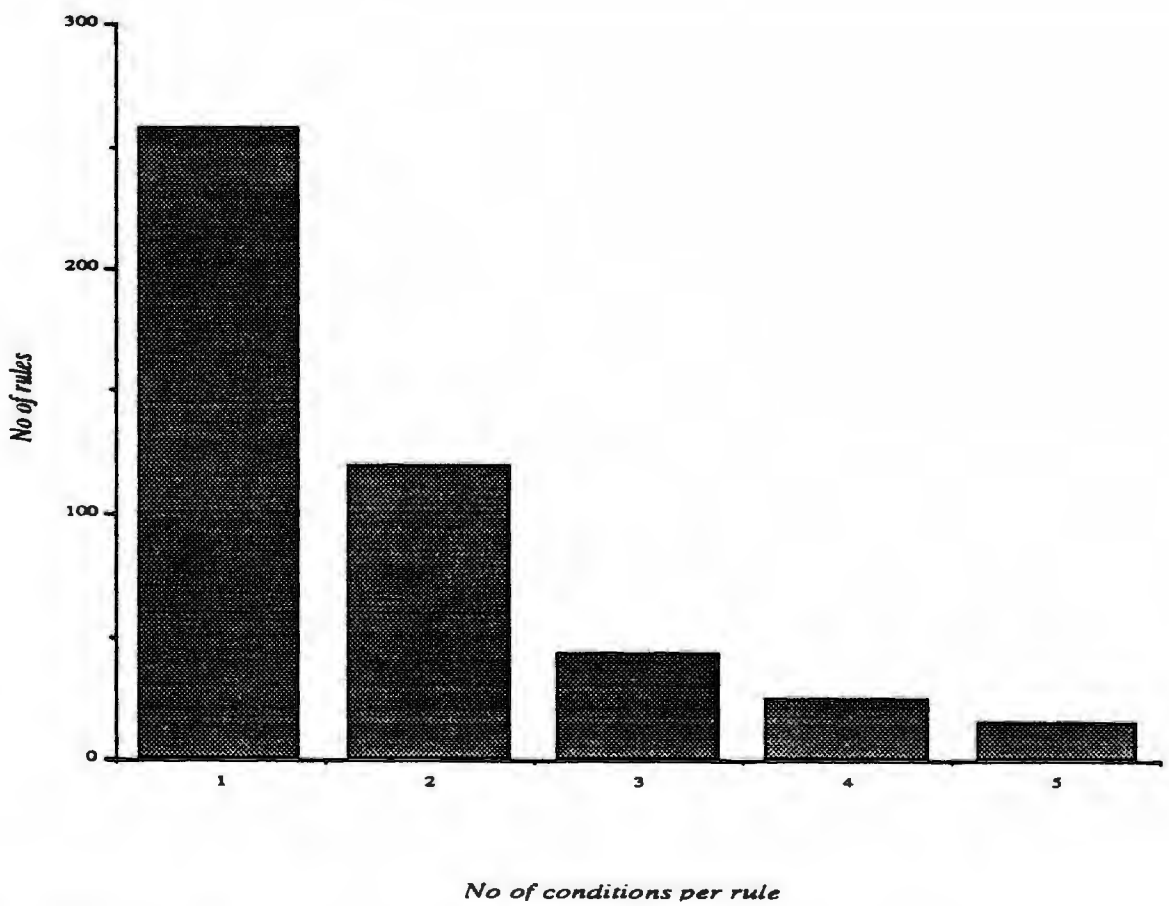


Fig 4 illustrates the complexity of rules in the St.Vincent's system. The number of rules with given numbers of conditions in the premises are shown. Rules are restricted to conjunctions of conditions. Only the first 500 rules in the system are shown

Fig 1 shows a ripple down rule knowledge base as a binary tree, however this should not be viewed as a conventional tree. Firstly each node contains a rule. The complexity of the rules is shown in Fig 4. Rules to date have up to 5 conditions. The majority of the rules have only a single condition. This does not imply that the knowledge and knowledge base are simple. A rule in a conventional expert system is equivalent to a ripple down pathway from root to leaf. The number of satisfied rules along pathways is shown in Fig 5. To be fully equivalent to a conventional rule both the positive and negative branches along a pathway need to be taken into account, indicating that the knowledge in ripple down rules is fairly complex. Figs 4 and 5 also indicate the advantage of ripple down rules. Most of the individual rules added are very simple, for in context it is simple to identify the one or two key conditions that discriminate between the old incorrect interpretation and the correct interpretation. Ripple down rules are very general and un-encumbered by the inclusion of the all the conditions required to make the rule work in a conventional expert system.

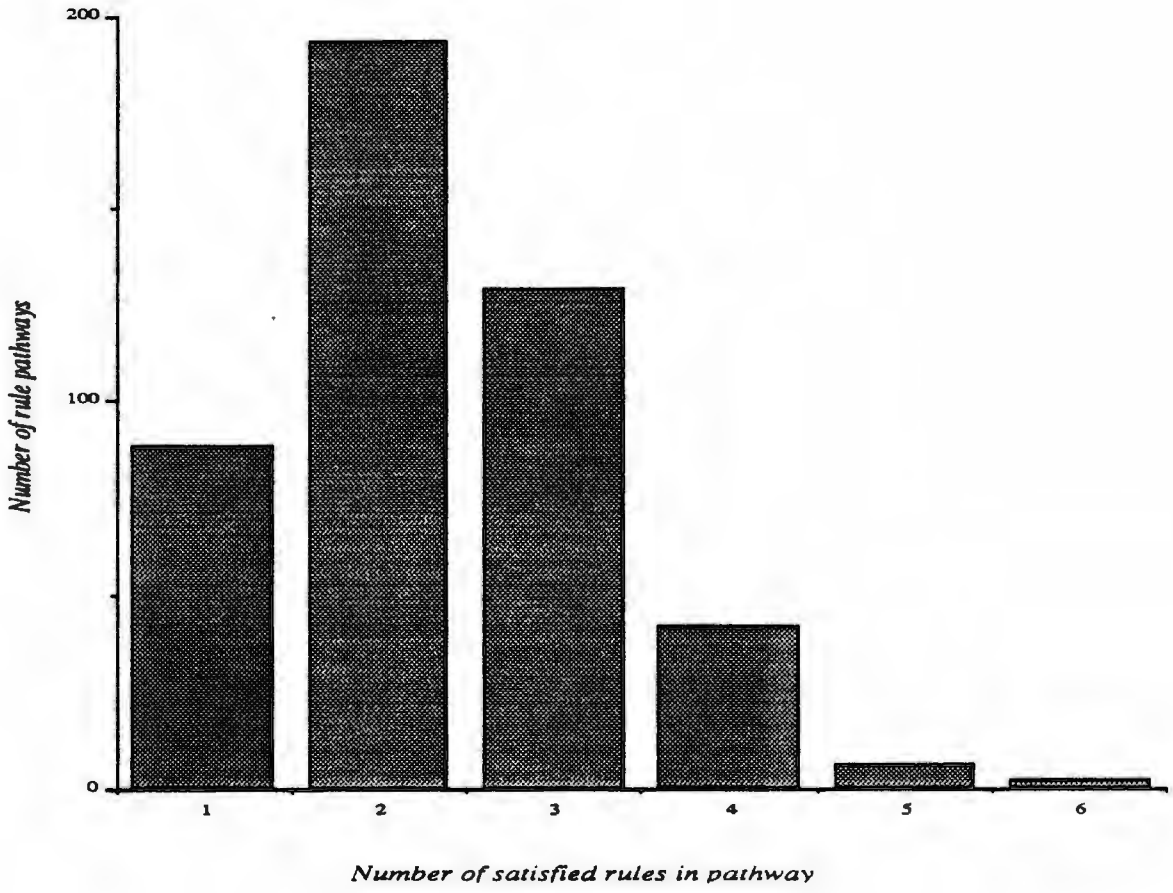


Fig 5 illustrates the the tree structure for the St.Vincent's system. The number of rule pathways of various lengths are shown. Only satisfied rules in a pathway are shown. Pathways are much longer if all rules traversed are included. Only the first 500 rules are shown.

An important feature of fig 5 is shown differently in Fig 6. These figures indicate that there are not many correction rules added to earlier rules. Fig 6 indicates that the knowledge base could be viewed as a conventional flat expert system with corrections added to rules rather than as a tree. The difference from a conventional expert system is that as soon as a rule is fired only rules that are corrections to that rule can fire and the traversal down the left most branch, equivalent to the flat expert system is left behind. The longest branch is the one where all rules have failed. This indicates that although experts give knowledge in context so that ripple down rules are required, they do make reasonably good rules so that often a new rule is required rather than a correction. With hindsight this must be true otherwise conventional expert systems would not work at all. In reality they obviously do work but are difficult to build and maintain on a large scale, because knowledge is taken out of context. The fact that we can communicate at all means that we do generate reasonably good models, reasonably good hypotheses of the world, but just as all hypotheses can be falsified because they are only true in a particular albeit fairly general context, so too, rules no matter how apparently general require further refinement in context.

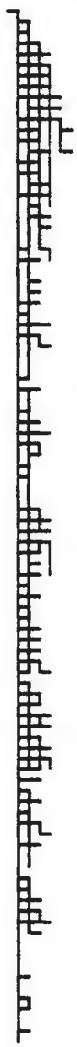
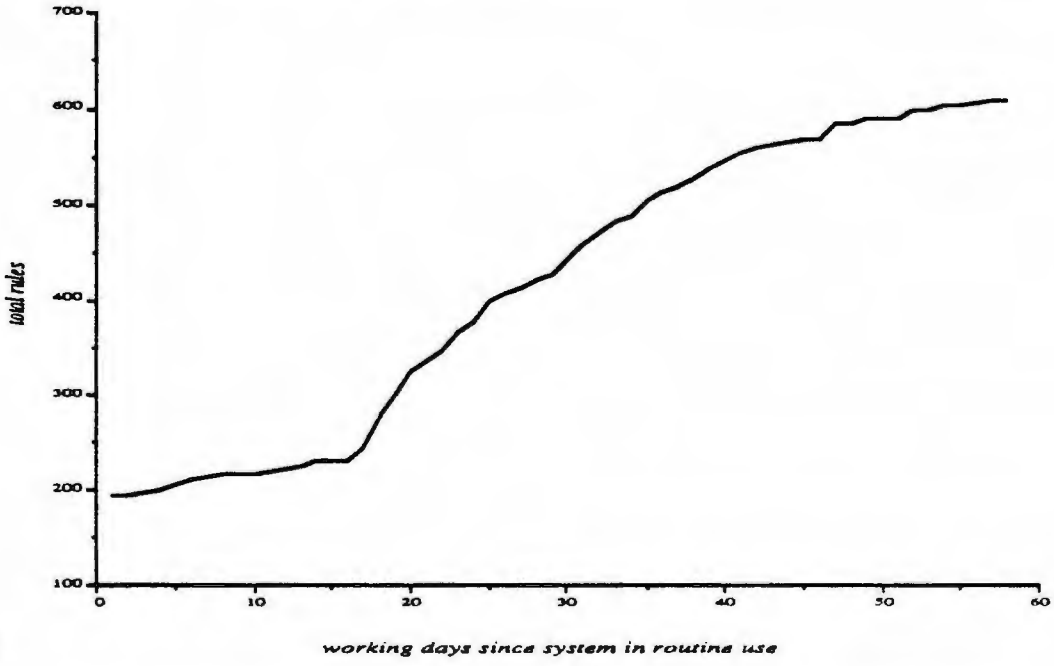


Fig 6 shows the entire ripple down rule tree whereas Fig 5 indicates only the number of satisfied rules in a pathway. False branches are vertical down the page and true or satisfied branches, horizontal. In contrast to normal tree diagrams all branches are the same length which results in parts of the tree being overwritten. However this representation accurately indicates the number of satisfied rules and unsatisfied rules in any pathway, but because of the overlay the particular sequence of satisfied and unsatisfied rules is obscured. This figure illustrates that the tree is extremely unbalanced. The first half of the tree going from top to bottom is mainly thyroid rules while the remainder of the tree is mainly blood gas rules.

At the time of writing this system has been in routine use for about eleven weeks and so far and just over 600 rules have been added. Development commenced with thyroid data and is now mainly concerned with acid/base balance. The system also has some catecholamine and diabetes rules. The thyroid development was completely separate from the previous Garvan system to provide a further evaluation of the method. No knowledge from GARVAN-ES1 was used and the knowledge was added by an expert who was not involved in the original development (or any other expert system project). Fig 7 illustrates the rate of growth of the knowledge base. (Note that Fig 7 shows the full 600 rules but figs 4, 5, 6 & 8 the first 500). At about 230 rules, the thyroid component of the system is about 95% correct which is consistent with earlier Garvan results and suggests that it will at least double in size (Compton and Jansen, 1990a). The Garvan system was 96% correct when introduced into routine use, however it doubled in size before reaching 99.7% acceptance by experts. In the redevelopment of GARVAN-ES1 as a ripple down rule system 550 rules were required to reach approximately 99% accuracy. The rate of thyroid knowledge acquisition has slowed down to a rule every few days, so that the system seems to be in the maintenance phase. Note that on Garvan archival data about 72% of the results are normal so that an interpretation level of over 95% represents a significant development. We are not sure what to expect with the long term development of this system. The expert may eventually decide that further changes are not warranted, but since the knowledge acquisition is so easy, the system may be continually refined, incorporating new tests, new styles of interpretation, the latest fad as it arises. This is what experts do but is it what an expert system should



do?

Fig 7 indicates the growth of the knowledge base. The x axis indicates the number of working days the system has been under development, about 11 weeks. The system started with approximately 200 rules because the expert was developing rules off line while interface problems with the main reporting programs were ironed out. However these rules were developed without the help of a knowledge engineer and were produced by the same process of identifying cases that required interpretation at report signing and then running them on the expert system by hand. The steep rise in the curve indicates when acid base knowledge base was commenced. The graph also suggests that the rate at which knolwedge is being added for the current domains is levelling out. Note that this is the only figure which covers the 600 rules current at the time of writing. Figs 4, 5, 6 & 8 illustrate the first 500 rules.

The rate at which acid/base rules are added also appears to be starting to slow down (Fig 7). The expert is at present is deciding what domain to tackle next. These are still very preliminary results, but it appears clear that the ripple down rule approach is viable in the lab environment and that such systems can be built by experts without knowledge engineering help. Since the system went into routine use there have been no queries in any form to knowledge engineers and there have been no knowledge engineers available at St.Vincent's. The only queries have been from knowledge engineers to the experts wondering how the system is progressing. It should also be noted that the expert has no particular computer skills or interest beyond using simple wordprocessing, statistics and drawing packages as part of his normal chemical pathology work. The expert adds new rules to the system in response to picking up incorrectly interpreted reports when signing out reports, which is one of normal duties. It remains an expert task to sign all reports which are issued by the laboratory, providing a perfect opportunity to check interpretations and add rules if necessary. A number of experts sign reports but all reports that are misinterpreted are currently given to the one expert to add new rules to the system. Note also that the expert has his normal duties to fulfil.

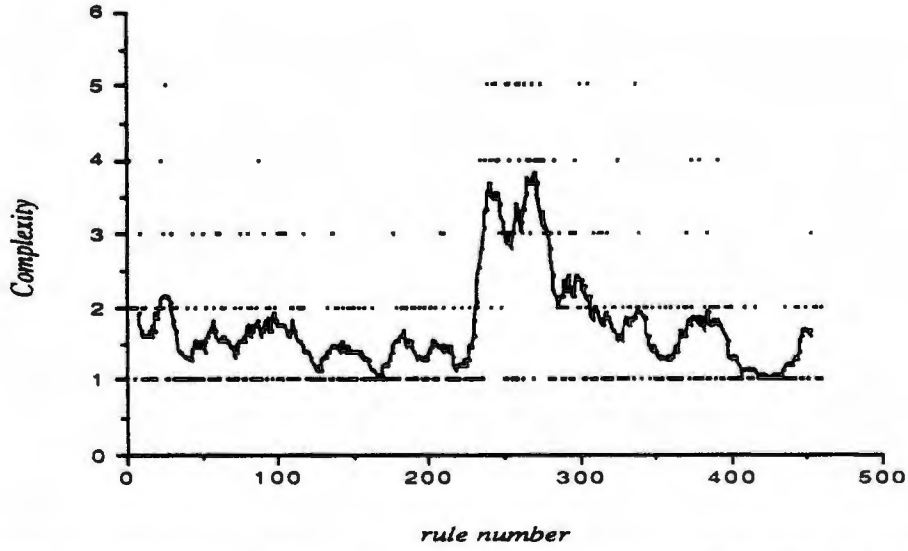


Fig 8 indicates the number of conditions in each rule against rule number. The dots indicate the actual number of conditions in each rule while the continuous line indicates a smoothed average of the number of conditions per rule. The sharp rise at about rule 230 indicates the commencement of acid base knowledge.

New areas can be opened up by simply adding a rule to give an interpretation for the new area. However once the new rule is entered the expert or experts must be able to follow up all misinterpretations and add correcting rules. This seems to be a manageable task in that a single expert has coped with adding rules as required for the high volume acid/base area while continuing with his normal duties. He has noted that the major problem, has not been creating the rule but deciding on the most helpful form of interpretation to provide. Different subdomains have different requirements for rules. Fig 8 illustrates that the initial rules added for the blood gas domain are much more complex than those added for thyroid data. However fairly soon the complexity of the blood gas rules dropped to the level of the thyroid rules. One would expect refinement rules to be more simple than the initial rules developed, if for no other reason than conditions have been "used up" in earlier rules in the path.

Other versions of "ripple down rules" have been developed in Prolog and in Hypercard. Brian Gaines has developed a further version in a knowledge representation server based on a term subsumption language (Gaines, 1991)

RIPPLE DOWN RULE LIMITATIONS

Repetitious knowledge acquisition

The most obvious problem of ripple down rules is the likelihood of repetitious knowledge acquisition. It is obvious that a ripple down rule system may end up with knowledge repeated in many places throughout the tree. We have not yet evaluated the St.Vincent's data but with the Garvan ripple down rule development repetition was not a significant problem, and there appeared to be no more than 13% of the knowledge repeated. Because the knowledge acquisition is so easy this is a small additional task. Suggestions that repetitious knowledge will be a problem are normally allied with suggestions that the tree should be periodically reorganised as it may be come very messy and unbalanced. In fact it seems that the tree's lack of balance shown in Fig 6 comes from the same cause as the small amount of repetition observed. Although experts' rules do need correcting they are fairly good rules so Fig 6 indicates more rules are constructed because the system fails to give a diagnosis, (the left most branch) than because rules need corrections. However because the acquisition is so simple the rules tend to be simple and general (Fig 4). The expert is able to make up the most general rule he or she wishes without having to add conditions because of the engineering requirements of trying to make the rule work. Thus Figs 4 and Fig 6 suggest that ripple down rules provide very simple general rules which require a fairly small number of corrections, hence the repetition is fairly small.

We have also investigated machine learning methods such as ID3 (Mansuri et al., 1991) since they produce a more optimal tree. For training sets of 8000 and 9445 case we found similar sized trees to the ripple down rule tree for ID3 and C4.5.(Mansuri, Compton et al., 1991) Brian Gaines carried out similar studies using his Induct algorithm and produced slightly smaller trees (personal communication). The use of C4.5 was not optimised so that it too may produce smaller trees. Also the Garvan ripple down rule system which was used as the basis of comparison was not complete and still had a 1.1% error rate. Inductive systems may thus produce more compact trees than the ripple down rules although these studies suggest that the difference will not be major e.g. 394 rules for Induct versus 550 for incomplete ripples. Even a two or three fold difference would be acceptable in terms of the ease of building ripple down rules. These studies also indicated an important advantage of ripple down rules. The Garvan ripple down rules were built by sequentially going through a large set of cases and making up new rules for cases which were misinterpreted by the rules already in the system. When an inductive method (ID3) was applied to the same data, 6000 cases were required in the training set before the inductive system performance caught up to the ripple down rules (Mansuri, Compton et al., 1991). For small numbers of cases ripple down rules outperformed the inductive system by a large margin. These results are not surprising, Gaines showed that the inclusion of rules provided by an expert greatly reduces the requirement for training cases for inductive learning (Gaines, 1989). This was also a difficult task as some 60 different classification of the data were required. These results do not imply that ripple down rules are "better" than inductive methods. If large data bases of training cases are available, inductive methods are obviously easier, however if these training cases are not available ripple down rules appear to produce a knowledge base not too much larger than inductive methods working on large training sets. Sooner or later repetitious knowledge acquisition will be a problem, but it is not a significant problem yet. A possible solution is outlined below.

The suitability of a training set for inductive methods does not depend just on the numbers of training cases, but on how well classified the cases are. This is a major problem in the domain of pathology interpretation. Ripple down rules allow the expert to construct free text interpretations, as the expert would do if adding an interpretation manually. It is very difficult to assign these interpretations to specific classes. Even if the expert is provided with tools to check the existing interpretations to decide which of these are being repeated there is no guarantee he or she will bother to do this carefully and not just resort to the addition of a new classification. If one wishes to produce a primitive system it may be possible to get the expert to provide coarse classifications, but expert interpretations are normally very subtle - GARVAN-ES1 produced 60 interpretations for example. We are currently working on tools to assist experts in identifying the relationships between interpretations. The aim of this work is to facilitate exploration of the knowledge base for educational purposes, it is not required for ripple down rules. To build a training set for inductive learning the expert would have to accurately classify thousands of cases. With ripple down rules, the expert has to add rules only periodically and the system can be introduced into routine use immediately.

Multiple classifications

A more important problem than repetitious knowledge acquisition is that multiple classifications may be required, for example if a patient has multiple independent diseases. At present the ripple down rule approach provides a single interpretation, although that interpretation may contain a number of parts. This may lead to large portions of the tree being repeated. The obvious solution of producing separate trees is not attractive because the domains may not be clearly separated. Data from one domain may be used in another and the artificial separation into sub-domains may be too crude resulting in the recurrence of the problem within the separate trees.

Another solution is to move away from the tree representation. We have proposed an approach whereby an "in context" knowledge acquisition methodology may be applied to a flat expert system in which multiple rules could provide interpretations which would then allow for multiple classification and help to reduce repetitious knowledge problems (Compton, Yang et al. 1991). In this approach rules can be modified, but each case for which a modification is made is kept, resulting in perhaps more than one case per rule in contrast to the ripples down rules. When a rule is narrowed to exclude a case a difference list can be produced from the intersection of all the cases connected to the rule compared to the new case. Generalising a rule is more complex in that all the other cornerstone cases have to be considered in arriving at a list of conditions which will allow the new case to fire on the rule but prevent any other case firing. With ripple down rules the exert picks from a single list. Similarly here the expert should be presented with a single list of conditions to choose from. Concepts of generalisation and specialisation should be hidden and the expert only deal with the difference list. A simpler approach may be to allow the ripple down rule interpreter to backtrack and explore any false branches. This is a legitimate strategy because the false branches are don't care rather than false. A rule is made up by the expert and automatically added to the false branch of the rule because the case failed to satisfy the rule. However, the expert is not aware of this so that a rule made up and attached to a "false" or "don't care" branch may be perfectly appropriate for some cases which satisfy the rule. An advantage of this approach over the flat system is that generalisation in the flat system requires all cases to be considered. It may be possible just to consider cases associated with the rules at the same level in the tree, that is attached to the new rule only by false branches. This would not provide a rigorous validation of the rule but it may be sufficient.

The backtracking suggested will change the scope of the knowledge acquisition task but not its nature; the expert will still identify differences to develop rules to produce the right interpretations, but will deal with multiple interpretations rather than single interpretations and the difference list may be more complex. The selection of conditions will also indicate at which leaf a new rule should be added. We hope that if the system is built like this from the ground

up it will not greatly increase the load for the expert. A prototype ripple down rule system with backtracking is currently being evaluated. We propose to minimise the problem at present at St.Vincent's by judicious choice of initial domains for interpretations.

Feature extraction

A further area of research for this system is in feature extraction or data reduction. A common assumption with knowledge based systems is that the data is going to be presented to the system in terms of the conditions the expert used in rules. This is only the case when there is a person entering the data into the system. This problem is not normally focused on, because of the knowledge engineering problems. Ripple down rules solve the knowledge engineering problem and leave one confronted with the feature extraction problem. The problem is obvious in terms of pathology reports. The expert is able to look at the data and identify the appropriate classification for a report. However for the rule to apply to as many cases as possible, it must abstract from the individual features of the data in this case, and the rule for the abstraction must be known, so that it can be applied appropriately to other cases. We have proposed above that fairly simple functions such as maximum, minimum and average etc. together with the option of changing cut-off values in context are sufficient feature extractors for much of pathology; however, there will be limits to their applicability and it would be preferable for experts to be able to define their own feature extractors. The current system allows the expert to change cut-off values in context. The features are not redefined, the rule simple eliminates cases from that pathway which do not have values in the specified range. It may be better to actually redefine the feature, particularly if the feature is more complex than those used here . This new definition would then be taken out of the context to be applied globally *to all rules developed after this point.* The definition of the feature would not be changed for any earlier rules, only for rules still to be added. It remains to be seen whether this is a useful strategy. Menzies has suggested applying this approach more generally to all procedural knowledge (Menzies 1991). Ideally experts should be able to define features themselves. For complex features it would be more appropriate to use a programmer or knowledge engineer, but experts may be able to deal with simple feature extraction or minor extensions to complex feature extractors. Menzies has proposed that there should be layers to procedural knowledge and that meta-heuristics control the order in which this procedural knowledge is presented to the expert so that she or he will be encouraged to make changes at as high a level as possible. As with the current system the expert should be able to go down to the actual data level if required. These ideas are still speculateive and may or may not be fruitful.

The distinction we have made between feature extraction and other rules is not clear cut as feature extractors can be expressed in rules and are context dependent. The flat nature of the ripple down rules whereby each rule provides a classification provides a basis for a distinction. A feature can be defined as any aspect or abstraction of the data that is used in a rule to provide a final classification. A feature reduces the data in that a number of different data patterns may show the same feature and so using this feature in rules enables the rules to be more general. This distinction is not so clear cut in expert systems which use a number of levels of intermediate variables in their reasoning. The rules that define features will be context dependent and should be able to be refined in context in the same way as ripple down rules, but it is implicit that may have a more general scope than a particular context in the ripple down rule tree. The solution seems to be to maintain trees both for feature refinement and the knowledge used to reach final conclusions. Links are maintained between the trees so that newly defined or modified features are only used in rules added to the main tree after the feature has been changed. This suggestion remains to be evaluated.

It should be noted that feature extraction only emerges as a problem because the knowledge acquisition has been simplified. Feature extraction is normally inextricably interwoven with the problems of knowledge engineering. Feature extraction can be seen as different from attribute identification. In the type of domains we are considering the data to be used in the expert system has all been identified, generally because it is available on a computer. Feature

extraction is concerned with identifying the features of the data that experts wish to use in rules. Attribute identification is concerned with identifying what data is used by the expert. The attributes identified by repertory grid tools such KSSO (Gaines and Shaw 1990) will normally also be features in that they will normally be used directly in rules without further abstraction. The importance of feature extraction does not apply in cases where humans who can identify the features required enter data into the expert system.

RIPPLE DOWN RULE POSSIBILITIES

Ripple down rules make possible a particular type of expert system. The traditional expert system is normally set up as a once only effort with the hope that its expertise will be useful for a long time to come. Maintenance problems generally confound this hope. Repertory grid tools in contrast have been sometimes used for creating "disposable" knowledge bases. The development of the knowledge base is used to clarify and support decision making when complex and important decisions to be made. The 500 or 600 hundred knowledge bases at Boeing which seem to have been used and then discarded have been used in this way (John Boose, personal communication). Ripple down rules opens up the possibility of the evolving knowledge base. As knowledge changes further refinements are added to the knowledge base. As new sources of data are brought into play in decision making it takes a considerable time before experts fully explore how the data should be used and move away from earlier inadequate data. Ripple down rules provides a way of allowing the knowledge to gradually move with the changes. It also provides a way of tailoring knowledge to very specific local concerns - just add further refinement to the knowledge base. In the pathology area, knowledge bases may be passed from lab to lab but these will then be tailored to local preferences and it is probably more likely that they will be entirely local. A particular feature of ripple down rules is that maintenance time for each maintenance incident remains constant regardless of the size of the knowledge base.

A question arises with expert systems as to whether they perform well according to objective tests. This is not a problem for the expert system but for the expert. If the expert system accurately reflects the expert, then the question is whether the expert performs reliably. This distinction becomes more obvious with ripple down rules as the expert refine the system to his or her personal idiosyncrasies. The current expectation at St.Vincent's is that there will never be a final expert system for the interpretation of all of chemical pathology, but that it will constantly evolve as lab practice evolves. None of this diminishes its role in providing automated interpretations of laboratory reports to assist clinicians receiving the report, however, it evolves as does the pathologists expertise.

The current application areas of ripple down rules are for classification tasks where all the data required is available and where there is an expert or some suitable person able to detect errors the system makes in classifying cases. This final requirement is not onerous. If any change is to be made to a knowledge base it will be made in response to a case being miss-classified. This implies the existence of both the case and someone to detect the missclassification and so provides the basis for ripple down rules. Secondly any significant expert system will be tested on enough data to evaluate all aspects of its knowledge before being put into use and again the output will be checked by someone. Such a system could have been built using ripple down rules. The conventional approach to building an expert system is to build the knowledge base and then validate it. Ripple down rules combines the validate and knowledge acquisition processes.

It is likely that ripple down rules would also be suitable for problems requiring backtracking. Backtracking is used whenever one wishes to minimise data collection. The tree form of ripple down rules and the small number of conditions in most rules suggest that a fairly minimal number of conditions will be asked for using a ripple down rule strategy, but it may be possible to further reduce this. One always starts at the top most rule in using ripple down

rules. However in asking for data for this rule one could determine the order in which data is requested according to the frequency with which this condition appears in all the rules in the left most path of the tree, so that as many rules are eliminated as possible if the condition is not present in the data. If the top rule fails the process is repeated on the next rule down until a rule fires. Once a rule is found that fires the process is repeated one level down. This is only a preliminary suggestion which needs expansion to deal with ripple down rule with backtracking.

A final proposal is that ripple down rules may provide a particularly useful rule trace. A conventional rule trace provides little explanatory power because the experts knowledge is obscured by conditions added to rules for engineering reasons. With ripples the rule is exactly as expressed by experts. Further, each rule is a correction of an earlier rule and indicates the conditions under which the earlier rule does not apply. Each rule also has the case that promoted its inclusion stored. This may be an important educational resource, particularly in medicine where much of the training is case based. A pathway may also includes a number of rules which failed. The conditions in some of these rules may not contradict the conditions in the rules that were satisfied. These rules thus indicate conditions which if they were present would alter the classification being made and may also be used for training.

SUMMARY

It has been proposed that expert knowledge is always provided in context and should therefore be used in context. A knowledge acquisition methodology has been proposed to capture and use knowledge in context. In this method the only knowledge acquisition task, is for the expert to select from a list of conditions. The expert thus has a very restricted task and no input into the way the knowledge base is structured. This greatly simplifies the expert's task and he or she need understand nothing of the knowledge base structure. It also results in a constant acquisition time for each new piece of knowledge regardless of the size of the knowledge base. We have implemented a system based on this approach which is now in routine use in a pathology laboratory with knowledge added by experts without the intervention of a knowledge engineer. This approach seems to simplify problems such as handling temporal data, and dealing with probabilities. Some limitations remain such as dealing with multiple classifications and further research in these areas is under way.

ACKNOWLEDGEMENTS

This work has been supported in part by grants from the Australian Research Council and Digital Equipment Corporation.

REFERENCES

Clancey, W. (1989a). Viewing knowledge bases as qualitative models. *IEEE Expert* Summer: 9-23.

Clancey, W (1989b) The frame of reference problem in the design of intelligent machines. in *Architectures for Intelligence: the 22nd Carnegie Symposium on cognition.* Hillsdale: Lawrence Erlbaum Associates

Coiera, E. (1989). Intelligent patient monitoring. *Proceedings of the Fifth Australian Conference on Applications of Expert Systems*, Sydney, 23-43

Compton, P. (1989). Expert systems for the clinical interpretation of laboratory reports, in O. Van der Heiden , N. den Boer and J. Souverijn (Eds.), *Clinical Chemistry: An Overview. Proceedings of the 1987 International Congress of Clinical Chemistry*. New York, Plenum. 615-628.

Compton, P., Horn, R., Quinlan, R. and Lazarus, L. (1989). Maintaining an expert system, in J. R. Quinlan (Eds.), *Applications of Expert Systems*. London, Addison Wesley. 366-385.

Compton, P. J. and Jansen, R. (1990a). A philosophical basis for knowledge acquisition. *Knowledge Acquisition* 2: 241-257.

Compton, P. and Jansen, R. (1990b). Knowledge in context: A strategy for expert system maintenance, in C. Barter and M. Brooks (Eds.), *Proc AI 88*. Berlin, Springer-Verlag. 292-306.

Compton, P., Yang, W., Lee, M. and Jansen, B. (1991). Cornerstone cases in a dictionary approach to rule maintenance. *IJCAI Workshop on software engineering for knowledge-based systems*,

Dreyfus, H. and Dreyfus, S. (1988). Making a mind versus modelling the brain: artificial intelligence back at a branchpoin. *Daedalus* 117(Winter): 15-43.

Gaines, B. (1989). Knowledge acquisition: the continuum linking machine learning and experise transfer. *Proceedings of the third European workshop on knowledge acquisition for knowledge-based systems*, Paris, 90-101

Gaines, B. (1991a). Integrating rules in term subsumption knowledge representation servers. *Proceedings of AAAI-91*

Gaines, B. (1991b). Between Neuron, culture and logic: explicating the cognitive nexus. *Proceedings of ICO'91: Cognitive Science: Tools or the development of organisations*,

Gaines, B. and Shaw, M. (1990). Cognitive and Logical Foundations of Knowledge Acquisition. *AAAI Knowledge Acquisition Workshop*, Bannf,

Horn, K., Compton, P. J., Lazarus, L. and Quinlan, J. R. (1985). An expert system for the interpretation of thyroid assays in a clinical laboratory. *Aust Comput J* 17 : 7-11 .

Lonergan, B. (1959). *Insight*. London, Darton, Longman and Todd.

Mansuri, Y., Compton, P. and Sammut, C. (1991). A comparison of a manual knowledge acquisition method and an inductive learning method. *Australian workshop on knowledge acquisition for knowledge based systems*, Pokolbin, 114-132

Menzies, T. (1991). Concerning the use of procedural constructs as a knowledge acquisition technique. *Australian workshop on knowledge acquisition for knowledge based systems*, Pokolbin, 133-156

Popper, K. (1963). *Conjectures and refutations*. London, Routledge and Kegan Paul.

Shaw, M. (1988). Validation in a knowledge acquisition system with multiple experts. *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1259-1266

INDUCTION AND VISUALIZATION OF RULES WITH EXCEPTIONS

Brian R Gaines

Knowledge Science Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4
gaines@cpsc.ucalgary.ca

Abstract: The ripple down rule schema has been embedded within an integrated knowledge acquisition architecture such that: the induction module induces ripple down rules; the evaluation module checks them against large data bases, whether induced or manually generated/edited; the visualization module represents them graphically; and the knowledge representation server supports inferences based on them. In extending the system, ripple down rules have been fully integrated with the existing rule exception structure, and have themselves been extended to support multiple conclusions and multiple exceptions.

INTRODUCTION

There are now a number of significant examples of real-world applications of knowledge-based systems on a routine basis, and we are beginning to see the development of a second-generation literature that reports on longitudinal studies of practical experience, and on the rational reconstruction of working systems. Such reports are of great significance to the scientific advance of knowledge acquisition theories, methodologies and tools, since they provide a rich source of well-grounded experimental material. In particular, they present a challenge to ongoing research on knowledge acquisition to demonstrate that 'new' approaches are able to cope well with the data and applications involved, and, hopefully, provide insights and tools that can improve on what has already been achieved.

An important development in this area has been the dissemination of material by Paul Compton and various co-workers on the knowledge-based system for diagnosis of thyroid disorders developed at the Garvan Institute (Horn, Compton, Lazarus & Quinlan, 1985; Compton & Jansen, 1990a,b). The system is of particular interest because:

- It is in routine clinical use.
- Data on some 20,000 patients is available.
- The data is, in major part numeric, giving scope for inductive estimation of critical values.
- Conversion to symbolic form is available.
- The original rule base developed is available.
- An improved rule base using 'ripple-down' rules is available.
- Data from this system has already been used in a number of knowledge-based system studies.

This paper reports on a group of studies which took this dataset as a challenge to the integrated knowledge acquisition architecture described at last year's meeting (Gaines, 1990) and attempted to:

- Compare machine induction of ripple-down rules with human generation of such rules.
- Achieve adequate speed in both induction and evaluation on large datasets to support collaborative development of the knowledge base through machine support of human expertise.
- Provide visualization of ripple-down rules as a natural extension to a graphic knowledge editor.
- Integrate the ripple-down rule technique with 'rules with exceptions schema' and extend it to cope with multiple conclusions and missing data ('open' cases).
- Develop a logical analysis of ripple-down rules and rules with exceptions.

RIPPLE-DOWN RULES

Compton and Jansen (1990a,b) introduced the ripple-down rule technique as an elegant methodology for the acquisition and maintenance of large rule-based systems. The underlying concept is that people cope with the acquisition and maintenance of complex knowledge structures by making incremental changes to them within a well-defined *context* such that the effect of changes is locally contained in a very well-defined manner. Standard production rule systems do not have this property. The modularity of the rules themselves is not reflected in a modularity of the consequences of changes to those rules. Small changes can lead, through complex interactions, to major effects, making the development and maintenance of rule-based systems far more difficult than it appears at first sight (Li, 1991).

One merit of frame-based techniques is that they provide means for managing rules by structuring them as 'demons' concerned only with entities that are instances of specific frames. The taxonomy of the frames then provides a way to conceptualize the activities of particular clusters of rules, and the relations between entities and their slot fillers provides a way to conceptualize the propagation of rules. Hence, it is of interest to compare solutions to the problem that modify the rule structure with those that manage the problem through frames.

The ripple-down rule technique creates a two-way dependency relation between rules such that rule activation is investigated only in the context of other rule activation. If the premise of a parent rule is true for a particular individual then, if has no dependents, its conclusion will be asserted for that individual. If, however, it has an 'if-true' dependent then that rule, and its dependents, will be tested, and the original conclusion will only be asserted if the premises of none of them are true for the entity. Conversely, if the premise of a parent rule is false for a particular individual, then not only will its conclusion not be asserted but also, if it has an 'if-false' dependent then it, and its dependents, will be tested.

Thus, ripple down rules form a binary tree with two sorts of link between parent and sibling. Figure 1 shows six of the top level rules in Compton's reconstruction of the Garvan ES-1 diagnostic system as a set of ripple-down rules. There are 61 possible diagnostic combinations, labeled 00 to 60. The top level rule, 0.00, is an overall default with no conditions stating that is no other rule is activated the diagnosis is 00, no thyroid disorder. This rule is always true so it has only an 'if-true' link to rule 1.46 which gives four conditions leading to diagnosis 46. If these conditions are not met, the 'if-false' link to rule 2.32 is followed. If the conditions are met then diagnosis 46 is not immediately indicated. The 'if-true' link to rule 497.48 is first tested and if its conditions are satisfied its diagnosis of 48 takes precedence over 46.

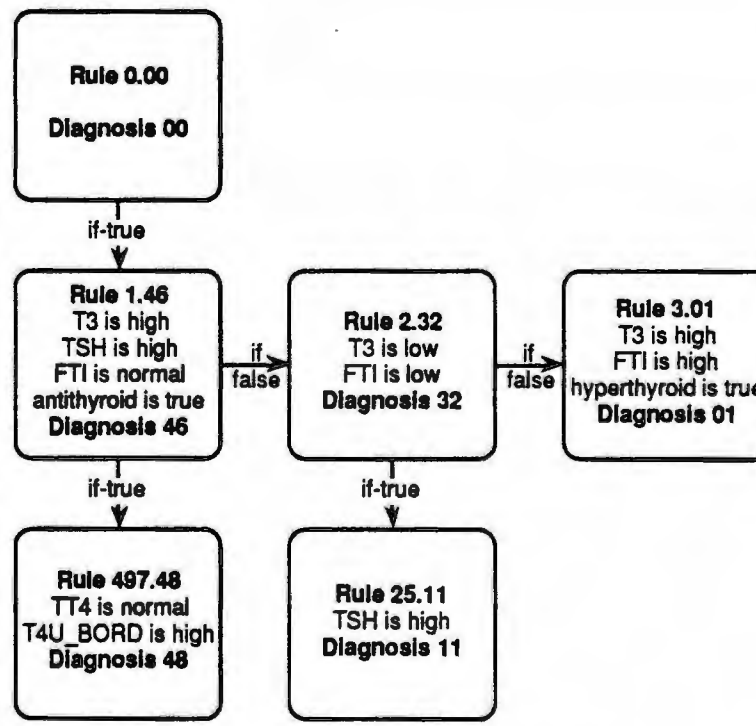


Figure 1 Some ripple-down rules

Logically, ripple down-down rules have exactly the same representational and deductive capabilities as standard production rules. Where they differ is in the strong context provided by the 'if-true' and 'if-false' links that guarantees that the effect of adding a rule is strictly contained in that it can affect the conclusions applied to cases previously falling under at most one other rule. A new sibling rule joined to a parent by an 'if-true' link can only change the conclusion applied to its entities falling under its parent, and no others. A new sibling rule joined to a parent by an 'if-false' link can only change the conclusion applied to its entities falling the rule with an 'if-true' link to the chain of which its parent is part, and no others. In general, a new rule can only affect cases falling under the rule responsible for the conclusion if the new rule were not present. This limits the comparisons and decisions required in knowledge acquisition and maintenance to the consideration of cases falling under only one rule.

For example, consider a situation in which a rule network has been built as shown in Figure 1, and a new case is under consideration for which rule 0.00 succeeds, 1.46 fails, 2.32 succeeds, and 25.11 succeeds. The conclusion is diagnosis 11, but suppose the correct diagnosis is 15. Since 25.11 succeeds but its conclusion is wrong, it is necessary to add a new rule with an 'if-true' link to 25.11. How should its premise be specified? Figure 2 gives the description of relevant cases available to the knowledge engineer. Case 25 is the one case that has so far been diagnosed using rule 25.11, and case 41 is the new case. In this table, the values of attributes satisfying the premises of the relevant rules are italicized, and the values in which two comparative cases differ are in bold. In considering the premises for a new rule, one is aiming to create one such that case 41 is included so that it can be diagnosed as 11, but case 25 is excluded so that it remains diagnosed as 11. Hence, the relevant differences are that *sick* is true for 41 and false for 25, *ovulatory* is false for 41 and true for 25, *TSH_BORD* is high for 41 and missing for 25. One could generate a very specify rule requiring all three differences to be present, or a more general rule requiring just one or two to be present. In this case the *sick* attribute was chosen as the differentiation resulting in the 'if-true' addition of rule 41.15 shown in Figure 3.

Now consider case 79 for which rule 0.00 succeeds, 1.46 fails, 2.32 succeeds, and 25.11 fails. The conclusion is diagnosis 32 from the last successful rule, but suppose the correct diagnosis is 01. Since 2.32 succeeds but its conclusion is wrong, it is necessary to add a new rule with an 'if-false' link to 25.11. How should its premise be specified? In figure 2, Case 2 is the one case that has so far been diagnosed using rule 2.32, and case 79 is the new case. One is aiming to create one such that case 79 is included so that it can be diagnosed as 01, but case 2 is excluded so that it remains diagnosed as 32. Hence, the relevant differences are that *hypothyroid* is true for 79 and false for 2, *ovulatory* is true for 79 and false for 2, *TSH* is missing for 79 and normal for 2, *T3_BORD* is missing for 79 and low for 2, *FTI_BORD* is low for 79 and missing for 2. In this case the *TSH, hypothyroid* attributes were chosen as the differentiation, and the *sick* attribute was also included resulting in the 'if-false' addition of rule 79.14 shown in Figure 3.

Note that the addition of a non-differentiating attribute causes no problems, providing at least one differentiating one is used. It is meaningful in that it will affect whether future cases fall under rule 79.14 or not.

This constrained but indeterminate choice of conditions for the premise of a new rule is the main feature of ripple down rule acquisition and maintenance. The expert, or knowledge engineer, has only to consider a limited and well-defined set of cases for which conclusions have been drawn. The possible clauses for the premise of a new rule are constrained to cover the new case but not the previous cases. Within these broad constraints, the expert or knowledge engineer has the discretion to use their knowledge to design as specific or as general a rule as they deem appropriate. Thus, the ripple down rule structure allows the computer to support the management of large rule bases while leaving the human designer the capability to use his or knowledge effectively in a well-defined decision framework.

	2	25	41	79
sex	female	female	female	female
on_t4	false	false	false	false
query_t4	false	false	false	false
antithyroid	false	false	false	false
sick	true	false	***true***	***true***
pregnant	false	false	false	false
I131	false	false	false	false
hypothyroid	false	false	false	**true**
hyperthyroid	false	false	false	false
lithium	false	false	false	false
tumour	false	false	false	false
goitre	false	false	false	false
hypopit	false	false	false	false
psych	false	false	false	false
antibs	false	false	false	false
surgery	false	false	false	false
treated	false	false	false	false
no_comment	true	true	false	false
screening	false	false	false	false
ovulatory	false	true	**false**	**true**
TSH	normal	*high*	*high*	***missing***
TSH_BORD	missing	missing	**high**	missing
T3	*low*	*low*	*low*	*low*
T3_BORD	low	missing	missing	**missing**
TT4	missing	missing	missing	missing
TT4_BORD	missing	missing	missing	missing
T4U	missing	missing	missing	missing
T4U_BORD	missing	missing	missing	missing
FTI	*low*	*low*	*low*	*low*
FTI_BORD	missing	missing	missing	**low**
TBG	missing	missing	missing	missing
TBG_BORD	missing	missing	missing	missing
diagnosis	32	11	15	14
RULE	2.32	25.11	41.15	79.14

Figure 2 Some cases relevant to these rules

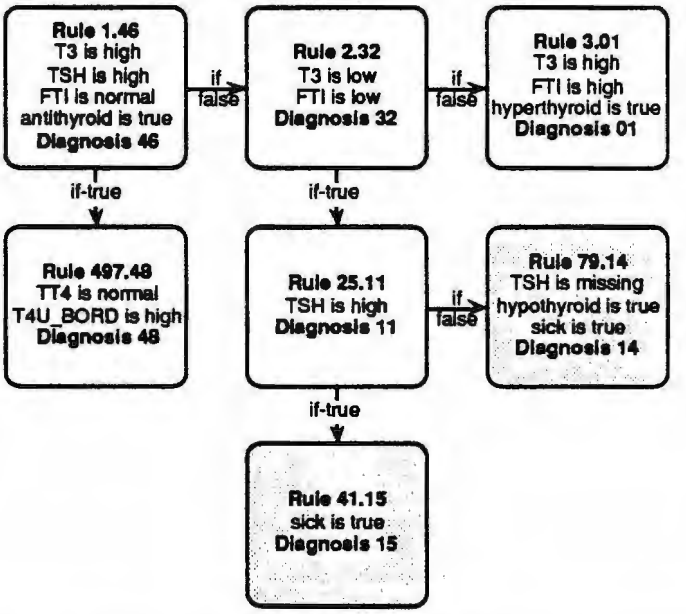


Figure 3 Ripple-down rules after incremental acquisition

There are three extensions to the situation illustrated that require further discussion. First, the new cases may be identical in features to an old case but have a different diagnosis. The cases cannot be differentiated and the expert or knowledge engineer has to cope with the indeterminacy, perhaps by gathering more data if there are missing values in the new case, perhaps by accepting the indeterminacy and giving alternative diagnoses, possibly with probability estimates. Clearly, this situation is outside the scope of a deductive process and requires a different solution to those under consideration.

Second, there are multiple cases falling under the existing rule, not just the single case so far exemplified. This causes no fundamental problems since, if the first situation does not apply, there will be at least one attribute differentiating the new case from any given previous case, and a combination of such differentiating attributes provides a rule differentiating it from all the previous cases. Hopefully, the situation simplifies, in that one attribute is sufficient to differentiate several previous cases, but even if it does not there is always a solution available.

Third, the expert or knowledge engineer may chose to not differentiate all previous cases in the multiple case situation. In this event an additional 'if-true' rule is required as a sibling of the new rule to cover the old cases that have been allowed to fall under the new rule. This would be done because the pair of rules generated in this way were perceived to be simpler than a single differentiating rule.

EXPERIENCE WITH RDRS AND DIRECTIONS FOR FURTHER RESEARCH

The utility of the ripple down rule approach has been demonstrated in large scale applications. The 650 complex rules of the Garvan ES-1 system have been reduced to some 550 simpler ripple down rules, and the rate of developing the rule base has been increased from about 2 rules a day to 10 rules an hour (Compton and Jansen, 1990b). Compton, Edwards, Lazarus, Malor, Menzies, Preston, Srinivasan & Sammut (1991) report a new successful application to chemical pathology at St Vincent's hospital.

The Garvan data were used some time ago for experiments in inducing rules using the system that was eventually issued as C4.5 (Quinlan, Compton, Horn & Lazarus, 1987), and Mansuri, Kim, Compton and Sammut (1991) have reported comparisons between ripple-down rules, ID3 and C4.5 on a study based on some 9,500 Garvan cases. In general, the number of rules generated by induction is similar to that generated manually through ripple down rules, as are the error rates. However, the induction tools used in these studies do not themselves generate the exception structure of ripple-down rules, and it would be useful to know how an induced set of ripple down rules compares with manually generated rules, in numbers, errors rates, and comprehensibility as knowledge structures.

The ripple down rule technique not only supports the incremental modification of the rule structure but also has a natural graphic knowledge representation as is apparent in Figures 1 and 3, and it would be useful to portray this in a knowledge visualization tool. This might aid understanding of the, still significantly large, knowledge structures generated, and also might support the global reconstruction of the ripple down rule tree, for example when over-specialization in earlier rules has caused a greater proliferation of later rules than is necessary.

As already noted, ripple down rules do not change the representational or deductive capabilities of the inference system. Hence, one now has three representational schema for rule-based systems: standard production rules without exceptions; rules with exceptions such that if the premise of one is satisfied it inhibits others (Gaines, 1991b); and ripple down rules with two forms of exception as described above. A multiplicity of representations may be valuable in knowledge representation in allowing a knowledge structure to be described in a way that is natural to human experts, or it may represent 'over-engineering.' Hence, it is interesting to investigate the relation between these three rule schema, particularly their inter-translation, and also to see whether they can be integrated into a single schema.

The ripple down rule schema as described depends on there being only one conclusion applicable to a particular case. It is often possible to restructure diagnostic situations such that this is so (about half the 61 Garvan diagnoses are compounds of multiple basic diagnoses), but this may be detrimental to natural representation and induction, and it is not always possible in more general systems. Hence it is interesting to investigate the extension of the ripple down rule schema to allow for multiple conclusions, and the possibility of embedding several such schema in a single knowledge structure concerned with multiple classifications.

These topics are the concern of the remainder of this paper. The ripple down rule schema has been embedded within the KSSn architecture (Gaines, 1991a) such that: the Induct module induces ripple down rules; the evaluation module rapidly checks them against large data bases, whether they are induced or manually generated/edited; the KDraw visualization tool represents them graphically; and the KRS knowledge representation server supports inferences based on them. In extending the system, the ripple down rules have been fully integrated with the existing rule exception structure, and have themselves been extended to support multiple conclusions and multiple exceptions. Since previous papers on KSSn have not focused on its inductive component, the next section will describe this independently of ripple-down rules.

KSSn INDUCTIVE MODELING SUB-SYSTEM

Inductive modeling of large databases is now an attractive proposition given the availability of fast empirical induction tools that can cope with the processing of large databases with many fields and data types (Piatetsky-Shapiro & Frawley, 1991). However, while the rules output from an induction tool may be capable of providing decision support when loaded into an expert system shell they have major weaknesses as 'knowledge', notably that they are difficult for people to understand, to some extent individually, and certainly in terms of their interactions.

An induction tool used on a database of a few thousand cases may sensibly generate several hundred rules in a few minutes. These rules may perform well in predicting one attribute from others but they will usually be almost impossible for someone to understand without extensive analysis. Full understanding probably only comes with editing and re-evaluation, "This rule makes more sense if I change it in this way. This block of rules is probably better thought of in this way. How am I doing with my 'improvements'?" There is a need for output modules that help people navigate through the results of empirical induction, seeing them in the most understandable form, and being able to edit and refine them to become humanly meaningful knowledge while continually checking that performance based on the revised rules is not deteriorating.

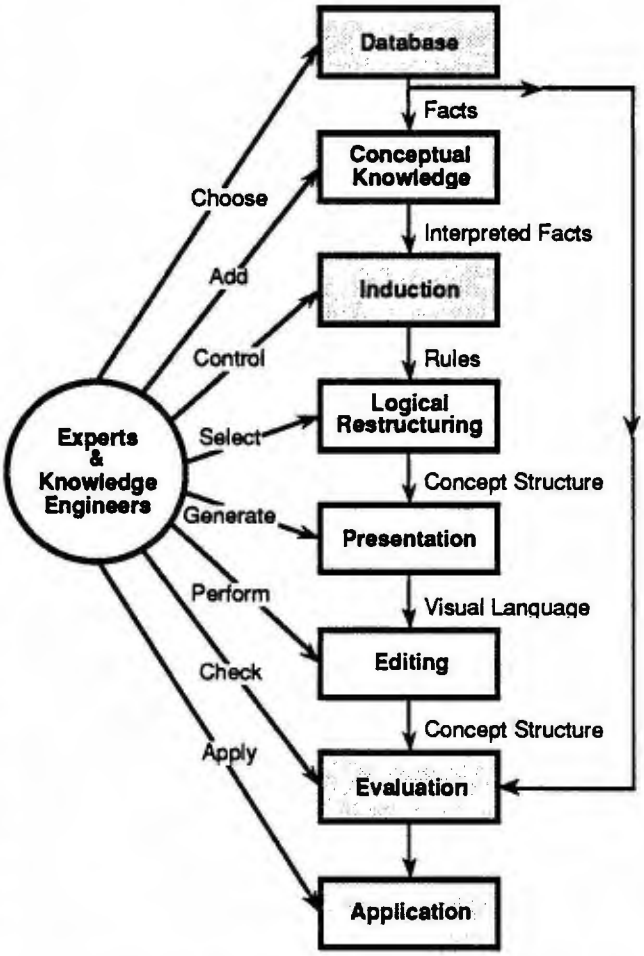


Figure 4 Induction process in KSSn

The design objectives for the induction functionality of KSSn are illustrated in Figure 4. A community of domain experts and knowledge engineers is attempting to extract knowledge from a large database. The 'community' may range from one individual to a large team, and its expertise may range from very little in an exploratory study to a great deal in a highly targeted analysis. The expertise may be assumed to change with the acquisition process so that, for example, highly automatic use of tools may be important in the early phases but detailed human control may become significant in later phases.

A stereotypical induction tool consists of the four shaded boxes in Figure 4:

Database→Induction→ Evaluation→Application

The additional interpolated boxes have been added to provide additional functionality to cope with some of the requirements noted above.

There is a need to supplement most databases with *conceptual knowledge* prior to induction because this is often implicit in the use of the database or just totally missing. However, its availability is important both to the effective use of the data available since the re-induction of already known concepts is wasteful of data, and detrimental to the quality of the results since the addition of a conceptual framework usually leads to increased generality and more meaningful interpretation. Often, the additional concepts involve ranges of numbers or grouping of categories. For example, in a medical database one may know that children in the age range 0-6 form a particular group at risk with respect to some diagnostic category and enter the conceptual classification that $0 \leq age \leq 6$ corresponds to "very young." This pre-classification will greatly reduce data requirements if it is appropriate (Quinlan, Compton, Horn & Lazarus, 1987). If it is not appropriate, given enough data, the induction algorithm will generate a more effective range. Núñez (1990) has shown that adding conceptual knowledge can be very effective in decision tree induction in producing more effective and smaller trees with less data. In general, introducing conceptual knowledge prior to induction is important in allowing the *data–knowledge* trade-off relation (Gaines, 1991d) to be manipulated effectively. It integrates the knowledge acquisition techniques of 'expertise transfer' and 'empirical induction'.

The *logical restructuring* of rules produced by an induction tool is also very important. The same logical structure can be manipulated to many different forms, some of which are very much more understandable than others. One example is the human tendency to conceptualize in terms of stratified systems of rules with multi-level exceptions (Compton & Jansen, 1990). For example, "a purchase order from a subsidiary to the main company should have order codes 08 or 09 except if that subsidiary is a manufacturing or packaging plant in which case the order code should be X5K or 113 unless the plant is overseas in which case the order code should be X5J." Such ramified structures of rules and exceptions are common in the constraints applied to large corporate database use, and their expression as an equivalent set of flat rules would be scarcely recognizable to users. Another example is the factoring out of common clauses in complex rules to form named sub-concepts, for example, "low quality supplier" for a supplier who is financially weak and has erratic delivery. Some degree of automatic restructuring is possible to minimize the expression of a rule set by using exceptions and factoring. However, minima are usually neither unique nor necessarily the most understandable expressions.

The *presentation* of the resultant concept structures is important to human understanding and evaluation as knowledge. Nosek and Roth (1990) have demonstrated empirically that the visual presentation of knowledge structures as semantic nets leads to more effective human understanding than does textual presentation of the same structures. Visual languages for knowledge representation can have well-defined formal semantics that are inter-translatable with standard textual representations (Gaines, 1991c). Hence, they may be treated as alternative ways of expressing a knowledge structure with, perhaps, the experts using one format and the knowledge engineers another.

Whatever form of presentation is adopted it is important that the users be able to *edit* in that format also. As already noted, some editing will inevitably be required and it is extremely frustrating to have to switch representations to make changes in the knowledge structures. Thus, a visual language should be implemented through a structure editor providing simple and natural interaction, not as a presentation mechanism only.

The results of editing should then be subject to *evaluation* as simply and directly as possible. For large datasets it may not be feasible to provide interactive evaluation in which the performance changes resulting from editing the knowledge structures are immediately displayed, but this is an important long-term objective as technology improves. Minimally, a fast evaluation against specified data with a display of the results in terms of the knowledge structure being edited is required. That is, the user should be able to see clearly and explore easily the sources of errors in terms of concepts, rules and cases, and their effects on overall performance parameters.

Figure 5 shows the architecture of an induction system developed to support the processes of Figure 4. The user interface for both the prior introduction of conceptual knowledge and the presentation and editing of the results of induction is a graphic knowledge editor for a term subsumption knowledge representation language (Gaines, 1991c). This enables the original database description to be expressed visually and then enhanced through the development of additional concepts and a concept hierarchy. The classification capabilities of the associated knowledge representation server are then used to extend the dataset with a set of binary attributes corresponding to a case falling under a concept or not.

The extended dataset is used as input to an induction engine that produces a minimal set of rules predicting specified attributes (which may include defined concepts). One option of the induction process is to produce stratified rules with multi-level exceptions (Gaines, 1991b). Biases may be introduced in the process through preferences for what values of attributes are taken to be defaults and what to be exceptions. The concepts forming the premises of rules can be factored to minimize the knowledge structure.

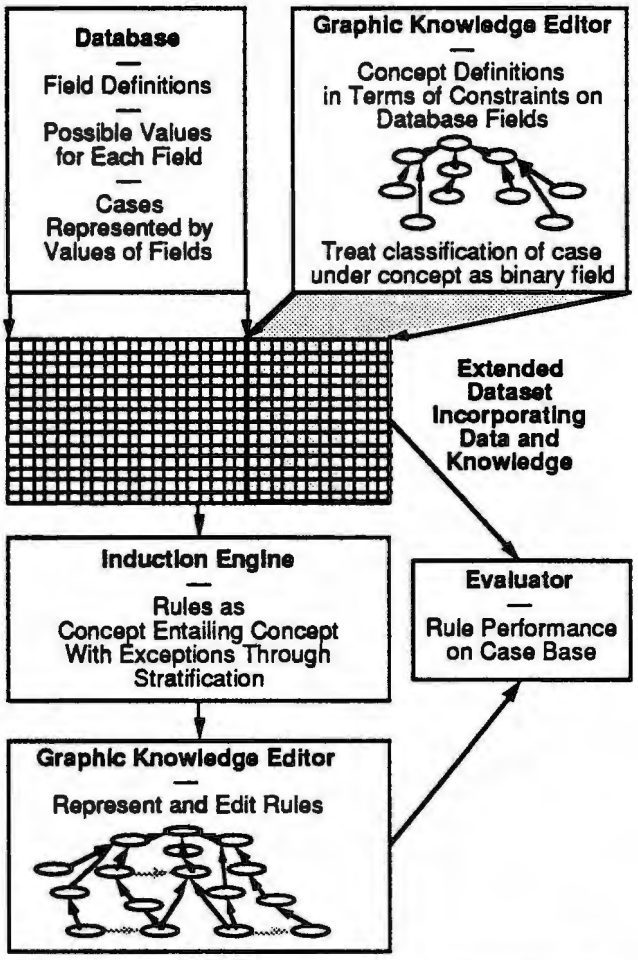


Figure 5 Induction system architecture

The resultant knowledge structure is presented visually in the graphic knowledge editor as an extension to the original structure entered. It may be freely edited and re-evaluated in terms of overall performance, and that of individual rules, against the original dataset and additional test data. Thus, from the perspective of the users, the flow of operations shown in Figure 4 can be carried out within the architecture of Figure 5. The components of Figure 5 are a set of modules around a knowledge representation server. This makes it feasible to integrate knowledge acquisition from databases with that from other sources such as text and human experts.

The system is best illustrated in operation through an example. Figure 6 is a dataset presented by Michie (1989) as a significant problem for NASA that was solved through automatic rule induction. The problem is one of deciding whether to use the shuttle autolander given various environmental conditions. The dashes in the dataset are *don't care* values. Their prevalence shows that it is a set of stereotypes rather than factual cases, and hence already contains additional knowledge. There would be 256 cases in the table if all the *don't care* values were instantiated. The elision of attributes in this way is a significant mechanism for the expression of expertise, as Bareiss (1989) showed in the PROTOS audiology study, and it important that an induction tool be capable of treating it correctly.

Case	stab	errors	sign	wind	mag	vis	class
1	-	-	-	-	-	no	use auto
2	no	-	-	-	-	yes	not auto
3	yes	lx	-	-	-	yes	not auto
4	yes	xl	-	-	-	yes	not auto
5	yes	mm	negative	tail	-	yes	not auto
6	-	-	-	-	out	yes	not auto
7	yes	ss	-	-	light	yes	use auto
8	yes	ss	-	-	med	yes	use auto
9	yes	ss	-	-	strong	yes	use auto
10	yes	mm	positive	head	light	yes	use auto
11	yes	mm	positive	head	med	yes	use auto
12	yes	mm	positive	tail	light	yes	use auto
13	yes	mm	positive	tail	med	yes	use auto
14	yes	mm	positive	head	strong	yes	not auto
15	yes	mm	positive	tail	strong	yes	use auto
16	yes	mm	negative	head	-	yes	not auto

Figure 6 Exemplary dataset for shuttle autolander decision (from Michie, 1989)

Figure 7 shows a set of rules derived by expressing as rules the decision tree given by Michie as a solution to the problem of predicting the final attribute "class" from the others. As Cendrowska (1987) has noted, the direct conversion of a decision tree to rules leads to some redundancy. However, those shown serve to exemplify the problem of comprehension when raw rule sets are presented to experts.

vis = no : *use auto*
vis = yes & errors = ss & stab = yes & mag = light: *use auto*
vis = yes & errors = ss & stab = yes & mag = med: *use auto*
vis = yes & errors = ss & stab = yes & mag = strong: *use auto*
vis = yes & errors = ss & stab = yes & mag = out: *not auto*
vis = yes & errors = ss & stab = no: *not auto*
vis = yes & errors = mm & stab = yes & sign = negative: *not auto*
vis = yes & errors = mm & stab = yes & sign = positive & mag = light: *use auto*
vis = yes & errors = mm & stab = yes & sign = positive & mag = light: *use auto*
vis = yes & errors = mm & stab = yes & sign = positive & mag = strong & wind = head: *not auto*
vis = yes & errors = mm & stab = yes & sign = positive & mag = strong & wind = tail: *use auto*
vis = yes & errors = mm & stab = yes & sign = positive & mag = out: *not auto*
vis = yes & errors = mm & stab = no: *not auto*
vis = yes & errors = lx: *not auto*
vis = yes & errors = xl: *not auto*

Figure 7 Decision tree for dataset converted to rules without pruning

Figure 8 shows a minimal rule set, equivalence in performance to that of Figure 7 generated by Induct from the data of Figure 6. One would expect the smaller rule set in itself to be easier to understand.

Representation of the subsumption involved in the exceptions is natural in graphical form. The lower part of Figure 9 shows the rules of Figure 8 translated directly to the visual language of the graphic knowledge editor in KSSn. In the upper part of the figure the database fields and possible values are shown as a domain description. Note that duplicate graphic items are treated as semantically identical so that the exception arrows between rules are shown only once.

```
: not auto
    Except:—
    vis = no: use auto
    stab = yes & errors = ss: use auto
        Except:—
        vis = yes & mag = out: not auto
    stab = yes & errors = mm & sign = positive: use auto
        Except:—
        vis = yes & mag = out: not auto
        vis = yes & wind = head & mag = strong: not auto
```

Figure 8 Induct rule set with exceptions

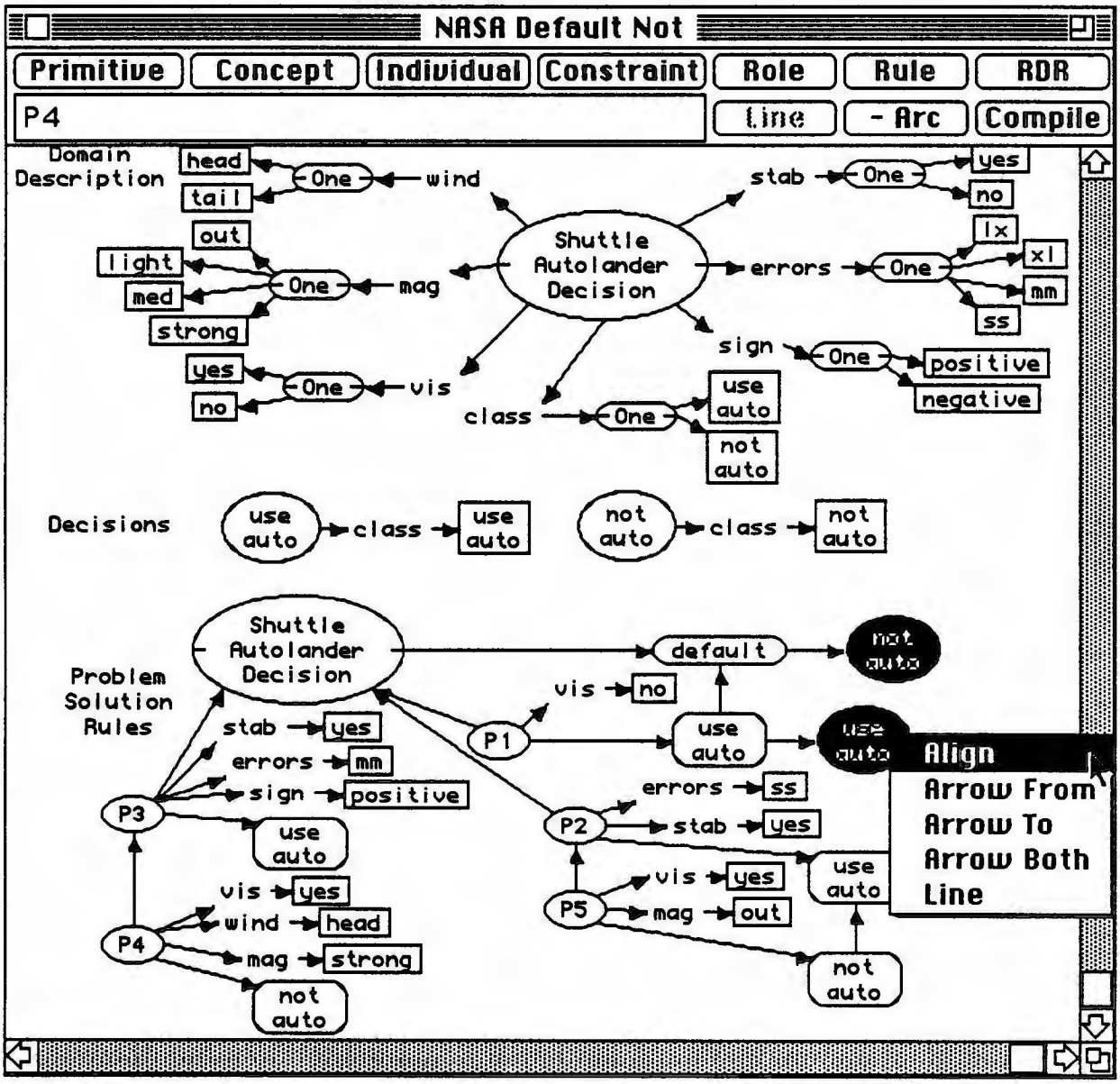


Figure 9 Database domain structure and rules in graphic knowledge editor

The visual language used in Figure 9 is precisely defined (Gaines & Shaw, 1990). Concepts are ovals, primitive concepts are ovals with small horizontal lines inside each side, individuals rectangles, roles unboxed text, rules rounded-corner boxes, constraint expressions rounded-corner boxes with small horizontal lines. Lines without arrows connecting primitive concepts denote that the concepts are disjoint, and connecting roles that they are inverse. The interpretation of the arrows in the editor is overloaded but well-defined by the types of the objects at their head and tail, e.g.:

concept → concept	definitional subsumption
concept → role → concept	definitional role with conceptual constraint
concept → role → constraint	definitional role with extension, cardinality or numeric constraint
constraint → individual	extensional constraint
individual → concept	asserted constraint on individual
individual → role → individual	asserted value of role for individual
concept → rule → concept	production rule
rule → rule	first rule exception to second

Human-computer interaction in the editor is modeled on Apple's *MacDraw* with additional features appropriate to the language such as arcs remaining attached to nodes when they are dragged. A popup menu as shown at the bottom right of Figure 9 allows connecting lines to be entered easily. The syntax of possible node interconnections and constraint expressions is enforced—it is not possible to enter a graph that is syntactically incorrect. Cut-and-paste of graphs and subgraphs is supported, and popup menus allow nodes to be connected with the minimum of effort. Updates are efficient and graphs with over a thousand nodes can be manipulated interactively. Scroll, zoom and fit-to-size facilities allow large data structures to be navigated easily. However, partitioning data structures over several screens is encouraged and has proved practical in managing large knowledge structures.

The user can evaluate the knowledge structure against specific cases visually or through automatic evaluation against cases stored in the knowledge representation server. He or she can compile the visual knowledge structures to the textual language of the server, or to a range of knowledge-based system shell languages, for evaluation on the original, or other, databases, and for application in a number of performance tools.

EXTENSION OF THE VISUAL LANGUAGE TO RIPPLE DOWN RULES

At first sight extending the visual language to encompass ripple down rules might seem obvious—add two arrows, one for 'if-true' and one for 'if-false', and then diagrams such as those of Figures 1 and 3 have a natural translation. The syntactic constraints are that: these arrows go only between rules; that each rule has at most one incoming arrow, and at most two outgoing arrows (one 'if-true' and the other 'if-false'); and that there are no loops. This approach leads to two questions: do either of the arrows correspond to the existing 'exception' arrow between rules; and does the syntax of multiple 'if-true' and 'if-false' arrows, into and out of rules have an unambiguous semantics capturing that of multiple conclusions and exceptions?

The semantics of the existing exception arrow between rules and that of the 'if-false' arrow in ripple down rules are the same. Their conceptualizations are different because the existing arrows are seen as inhibitors of the rules to which they connect, whereas the 'if-false' arrows are seen as passing on cases that do not fall under a rule. These two viewpoints are however complementary, and mutually consistent in the case of standard ripple down rules.

However, the extension of semantics based on two types of arrows to multiple 'if-true' and 'if-false' arrows entering and exiting rules breaks down. There is a natural disjunction for multiple 'if-true' arrows entering a rule—a case satisfying any of the rules initiating the arrows should be further tested. There is also a natural disjunction for multiple 'if-false' arrows entering a rule—a case satisfying any of the rules initiating the arrows should bot be further tested. However, if a mixture of arrows entered there is potential conflict and no natural resolution.

It turns out that the natural generalization is not to have two types of arrow, but rather two types of rule. One generalizes Compton's notion of *context* rather than that of 'if-true' and 'if-false' rules. The rule that an 'if-true' arrow enters marks the beginning of a new context established by the link from the rule that was true. All arrows from this rule propagate this context, and hence it is appropriate to mark the rule itself as being of a different type from the others. Figure 10 shows the rules of Figure entered in the grapher using only one arrow between rules but two different types of rule. An 'if-true' arrow is now one that enters a *context-marking* rule (with internal vertical bars at each end), hereafter termed an RDR. An arrow entering an ordinary rule is an 'if-false', or inhibitory connection. The semantics of multiple arrows in or out of rules is now unambiguously defined, and the only residual constraint is that there must be no loops.

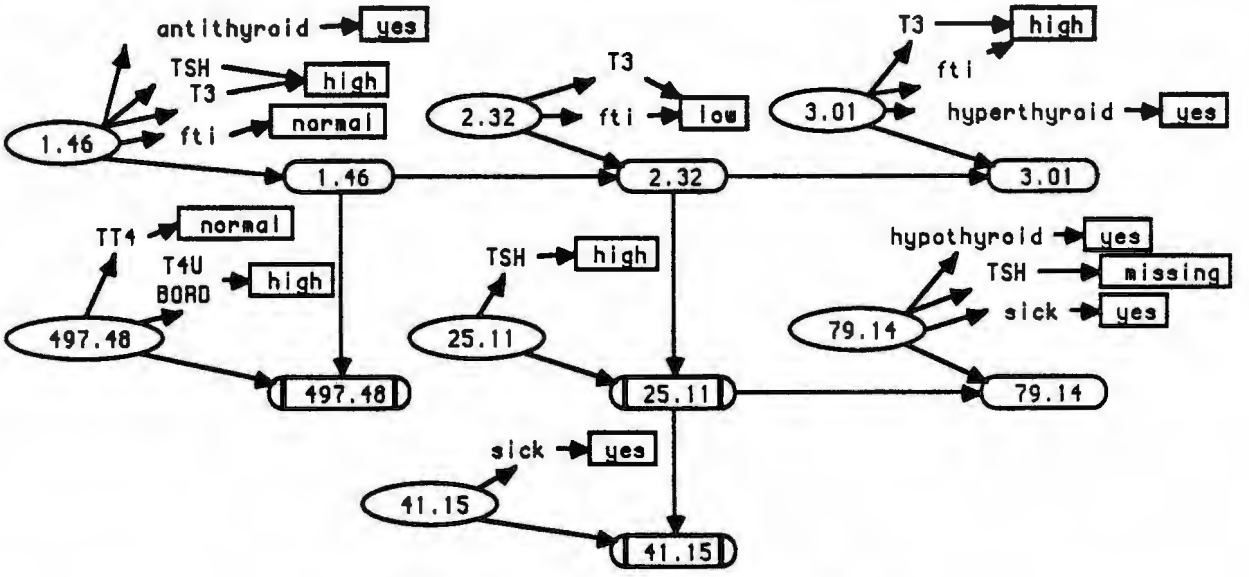


Figure 10 Ripple down rules of Figure 3 in KDraw

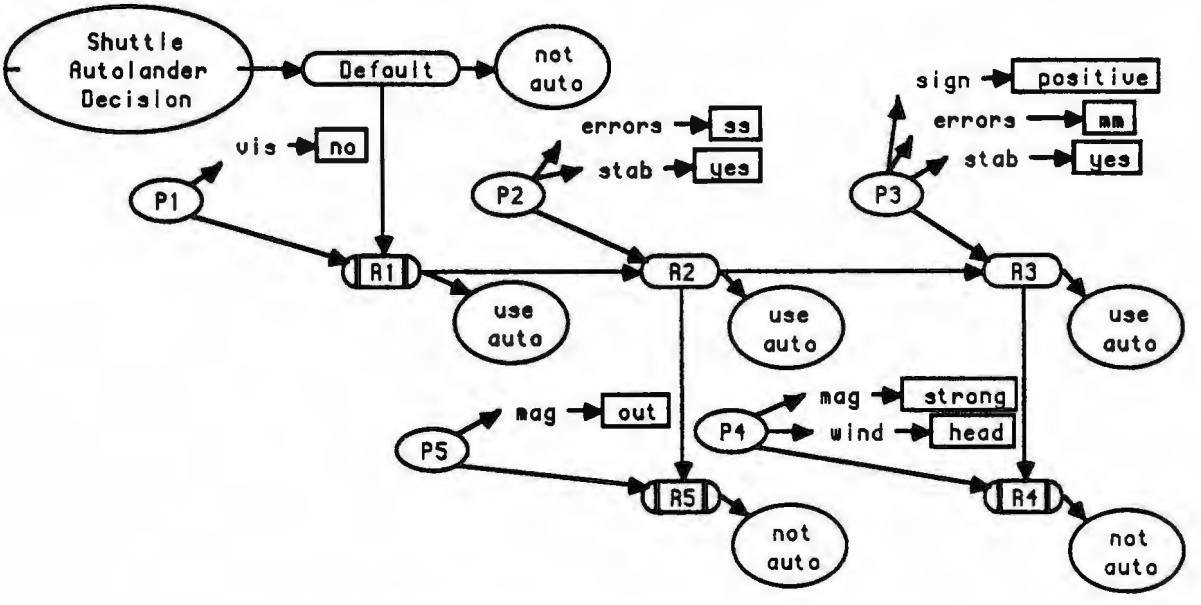


Figure 11 Ripple down rules for problem of Figure 6

Figure 11 shows ripple down rules induced by Induct for the shuttle autolander problem of Figure 6 with "not auto" as the default. In comparison with Figure 9, the subsumption structure of the premises is no longer overtly in the relations between the premise concepts, but is now in the relations between the rules. The structures are logically identical but their representational form is different. Figure 12 shows how the extended interpretation of the 'ripple down' arrows allows the solution to be formulated through 'if-true' arrows only, making the relation to Figures 9 and 11 apparent. Note the extra "vis = yes" clauses necessary in P4 and P5 that were implicit in the 'if-false' link from R1 in Figure 11. Note the additional 'if-true' link from R3 to the "mag = out" exception which was implicit in the 'if-false' link from R2 in Figure 11. The contexts are weaker in Figure 12 making the knowledge more explicit and the structure less compact.

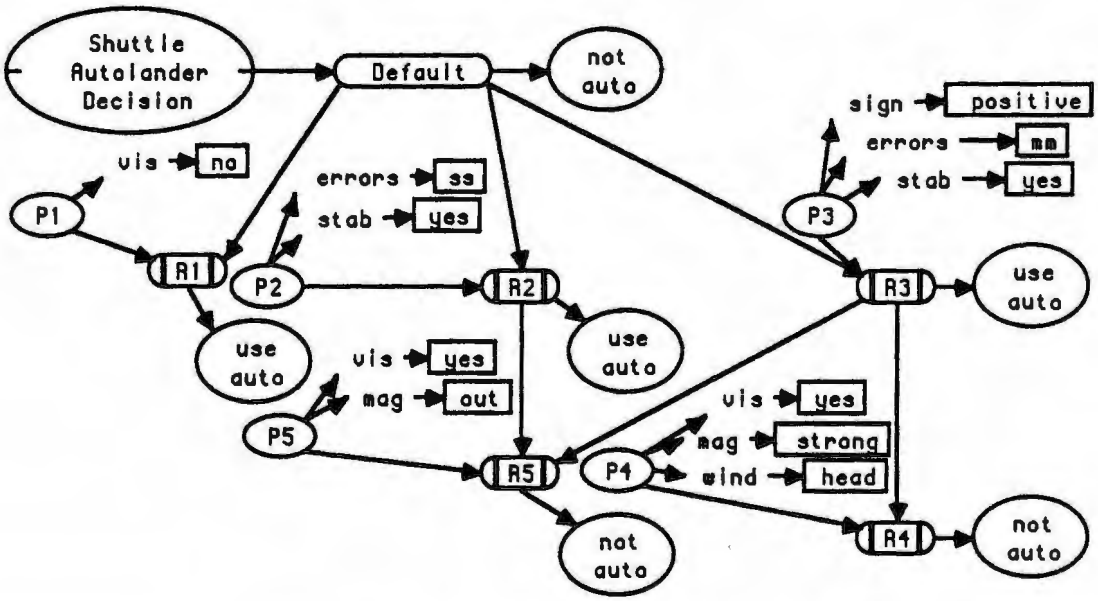


Figure 12 Variant rules for problem of Figure 6

Thus, as already noted, the two types of exception now allow a multiplicity of representations of the same knowledge structure. In Figure 9 there are only 'if-false', inhibitory links. In Figure 12 there are only 'if-true', excitatory links. In Figure 11 the two combine in a strict ripple down rule structure. Other combination are possible. Until there is further experience, probably of more complex situations with multiple conclusions possible for a single case, it is difficult to determine whether one has achieved 'representational richness' or degenerated into 'over engineering.' Currently, the naturality of KRS's original exception relation between rules, and the neat representation possible with ripple down rule structures as illustrated in Figures 10 and 11, suggests that the two forms of rule/link are useful knowledge constructs.

INDUCTION OF RIPPLE DOWN RULES

Inducing ripple down rules is very straightforward. The basic algorithm in Induct (Gaines, 1989) is to search for the premise for a given conclusion that is least likely to predict that conclusion by chance. Exception generation is automatic in that extending a premise with additional clauses to give fewer errors may also increase the probability that the improved result is due to chance—fewer cases predicted more accurately is not necessarily better on this measure than more cases predicted less accurately. When errors remain after the generation of a rule, Induct calls its rule generation algorithm recursively on the cases falling under the rule to generate exception rules for the error cases. This corresponds to the generation of the 'if-true' links in ripple down rules. To generate 'if-false' links, Induct calls the algorithm recursively for the residual cases that do not fall under the rule previously generated.

When Induct is run on the 9445 Garvan cases used in the UNSW studies of induction and ripple down rules, it generates 170 rules accounting perfectly for the dataset. This compares favorably with the some 550 ripple down rules generated in the reconstruction of the Garvan ES-1 using the same data. When Induct is run on the first 8000 cases and tested on the complete dataset it generates 172 rules and has an error rate of 0.4%. This compares favorably with the results reported by Mansuri, Kim, Compton and Sammut (1991) of: 550 rules, 0.5% errors for hand generated ripple down rules; 719 rules, 0.4% errors for ID3; and 515 rules, 2.1% errors for C4.5 with rule pruning.

Figure 13 shows the complete set of induced rules, diagnoses and number of cases falling under each rule. Apart from the difference in number, there is also a difference in structure from the manually produced set. The manual set has 41% 'if-true' links, and the mean number of clauses in a premise is 1.3. The induced set has 16% 'if-true' links, and the mean clauses is 3.8. It appears that rules in the manual set are over-general compared with those in the induced set and hence involve more correction rules.

It is reasonable to expect induction of ripple down rules to be substantially better than hand generation if we compare the circumstances of the human expert and the Induct algorithm. The expert and the algorithm are doing exactly the same thing at each node of the ripple down rule tree—that is adding additional rules to improve the performance. However, the expert is doing this in response to one erroneous case whereas the algorithm has available to it a complete analysis of the statistics of all relevant cases. Clearly, it is an empirical result as to how much better the algorithm will be, and the 3:1 improvement obtained for this dataset is only indicative of what might occur with other datasets.

There are a number of comments to be made on these results. First, that the decrease in the size of the rule set by no means detracts from the value of the manual acquisition of ripple down rules. At the start of a knowledge-based system development there may not be enough data available for induction, but there may be adequate human knowledge to build a working system. Later, when that system has been in use for some time, ripple down induction may be used to reduce the rule base without undermining the basic conceptual structures for use and maintenance to which those responsible for the system have become accustomed.

Second, there are more subtle reasons also why manual acquisition remains very important. These show up well in the recent St Vincent's studies reported by Compton, Edwards, Lazarus, Malor, Menzies, Preston, Srinivasan & Sammut (1991). The expert in entering new ripple down rules also enters new concepts. That is, the conceptual framework being applied to the cases is elaborated as part of the ripple down rule knowledge acquisition process. The contextual constraints already described continue to apply so that the use of new concepts can be managed locally and does not involve the expert in reanalysis of other parts of the rule tree. Some of the new concepts entered fall completely outside those already entered. Others are intervals on existing numeric data. The latter could be induced with sufficient data but the former could not. Other knowledge acquisition techniques, such as the repertory grid, might be used to elicit some of the additional concepts in advance, but there is already a strong case for manual ripple down rule elicitation as an effective acquisition technique for conceptual schema.

Third, the best features of the manual and inductive approaches can be combined. The evaluation tool in KSSn is fast enough for interactive use. It operates at some 200,000 rules a second on a Mac2ci and hence evaluates a rule set of some 170 rules on 9,500 cases in 8 seconds. An expert or knowledge engineer can make arbitrary changes to the ripple down rule structure and have a complete account of the effects rapidly enough to maintain reasonably interactive development. The Induct algorithm under the same circumstances takes 8 seconds on average to generate a new rule so that it can be used to advise the expert or knowledge engineer of an appropriate rule, which may then be manually amended, rather than generate all rules completely automatically.

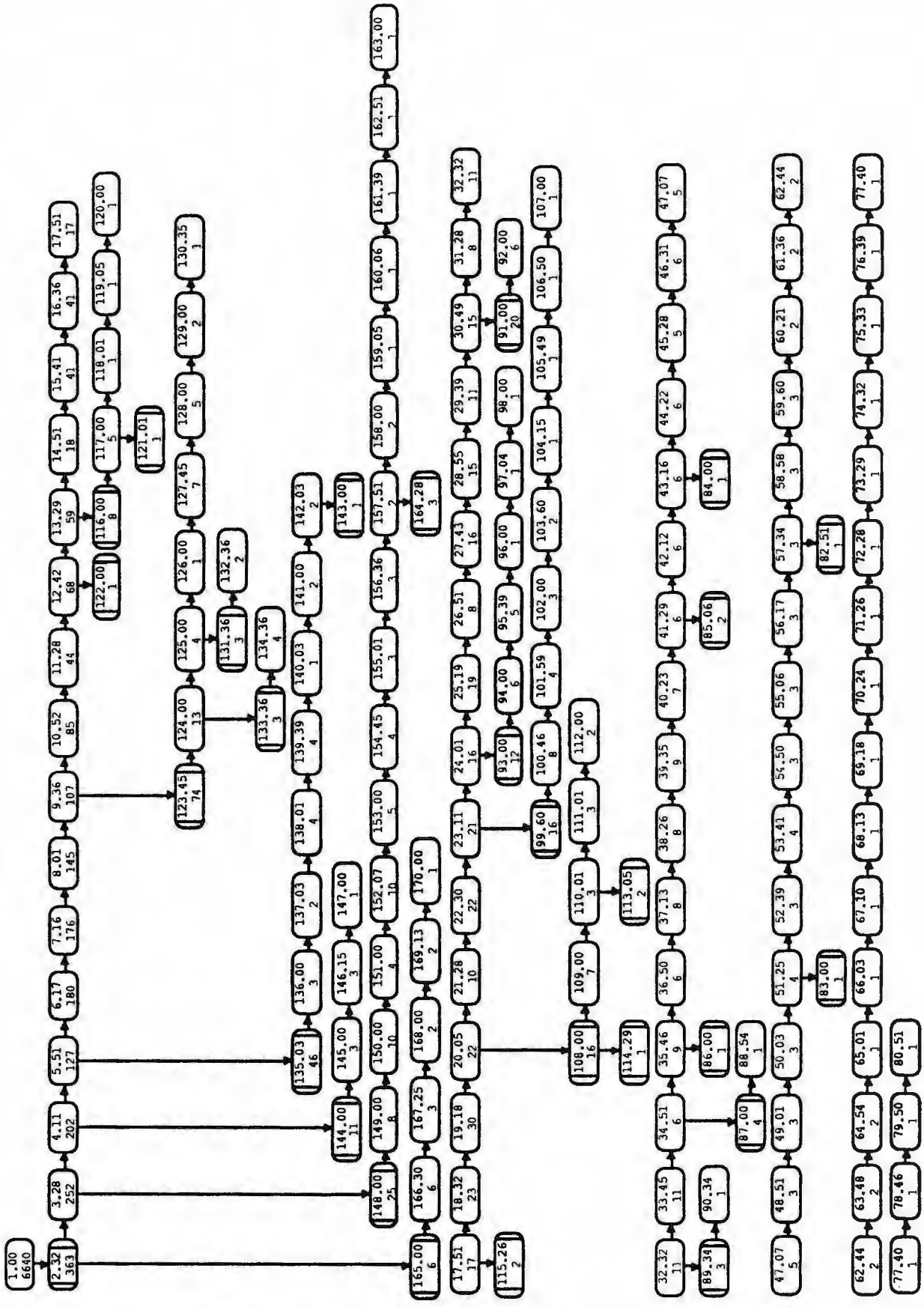


Figure 13 Ripple down rule structure induced for the Garvan data

CONCLUSIONS

This is a preliminary report on ongoing research. One can conclude so far that ripple down rules may be induced effectively and that they also integrate well with the visual representation of term subsumption knowledge representation schema. Extending and completing the system described to fully support mixed initiatives between manual and automatic knowledge modeling appears to offer significant advantages.

It will be interesting to determine whether the rule structure of Figure 13 is seen as a meaningful knowledge structure by those with relevant clinical experience. The top level structure in particular has large numbers of cases under rules with comparatively few exceptions and the rationale for each clause in the premise should be evident. Some of the exceptions correspond to the addition of diagnoses in a compound diagnosis which is clearly a valid reason for an exception. Others correspond to what may be variant diagnoses. The structure seems compact enough for a clinician to work through it methodically determining whether it is meaningful.

There is much that remains to be done with the Garvan dataset. Another 10,000 cases are available which will give some independent validation of the induced structures. There is adequate data to induce the numeric thresholds used in the conversion to symbolic form. It will be interesting to compare the detailed knowledge structures of the original rules, the manual ripple down rules and the induced rules. Paul Compton will be making the datasets and rules available as a Sisyphus project and it will be interesting to bring an increasing number of methodologies and tools within the same experimental framework.

It would also be very insightful to apply the same techniques to other datasets, notably the current St Vincent's study. It is not easy to find large-scale problems where both the data and expert knowledge are available to allow the types of methodologies and tools outlined in this paper to be properly evaluated. We need at least as much effort to be put into ongoing Sisyphus projects as is put into methodology and tool development. Knowledge acquisition needs to look to its empirical as well as its theoretical foundations.

ACKNOWLEDGEMENTS

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. I am very grateful to Paul Compton who provided the inspiration for much of this work. In particular, access to his own research and the datasets on which it has been tested have been invaluable in the work reported here.

REFERENCES

Bareiss, R. (1989). **Exemplar-Based Knowledge Acquisition.** New York: Academic Press.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Compton, P., Edwards, G., Kang, B., Malor, R., Menzies, T., Preston, P., Srinivasan, A. & Sammut, S. (1991) Ripple down rules: possibilities and limitations. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Sixth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.?. Banff (October).

Compton, P. & Jansen, R. (1990a) Knowledge in context: a strategy for expert system maintenance. Barter, C.J. & Brooks, M.J., Eds. **AI'88: 2nd Australian Joint Artificial Intelligence Conference, Adelaide Australia, November 1988, Proceedings.** pp.292-306. Berlin: Springer Verlag.

Compton, P. & Jansen, R. (1990b) A philosophical basis for knowledge acquisition. **Knowledge Acquisition 2**(3).241-258.

Gaines, B.R. (1989). An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. **Proceedings of 6th International Workshop on Machine Learning**, pp.156-159. San Mateo, California: Morgan Kaufmann (June).

Gaines, B.R. (1990). An architecture for integrated knowledge acquisition systems. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.8-1-8-22. Banff (November).

Gaines, B.R. (1991a). Empirical investigation of knowledge representation servers: design issues and applications experience with KRS. **AAAI Spring Symposium: Implemented Knowledge Representation and Reasoning Systems**. pp. 87-101. Stanford (March) and **SIGART Bulletin** 2(3), 45-56 (June)..

Gaines, B.R. (1991b). Integrating rules in term subsumption knowledge representation servers. **AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence**. pp.458-463. Menlo Park, California: AAAI Press.

Gaines, B.R. (1991c). An interactive visual language for term subsumption visual languages. **IJCAI'91: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence**. San Mateo, California: Morgan Kaufmann.

Gaines, B.R. (1991d). The tradeoff between knowledge and data in data acquisition. Piatetsky-Shapiro, G. & Frawley, W. **Knowledge Discovery in Databases**. Cambridge, Massachusetts: MIT Press.

Gaines, B.R. & Shaw, M.L.G. (1990) Cognitive and logical foundations of knowledge acquisition. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp. 9-1-9-25. Banff (November).

Horn, P.J., Compton, P.J., Lazarus, L., & Quinlan, J.R. (1985). An expert system for the interpretation of thyroid assays in a clinical laboratory. **Australian Computer Journal, 17**, 7-11.

Li, X. (1991) What's so bad about rule-based programming?. **IEEE Software** 103-105 (September).

Mansuri, Y., Compton, P. & Sammut, C. (1991) A comparison of a manual knowledge acquisition method and an inductive learning method. **Proceedings of the First Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.114-132. Pokolbin (July).

Michie, D. (1989) Problems of computer-aided concept formation. Quinlan, J.R., Ed. **Applications of Expert Systems Volume 2.** pp.310-333. Sydney: Addison-Wesley.

Nosek, J.T. & Roth, I. (1990) A comparison of formal knowledge representations as communication tools: predicate logic vs semantic network. **International Journal of Man-Machine Studies 33**, 227-239.

Núñez, M. (1990). Decision tree induction using domain knowledge. Wielinga, B., Boose, J.H. & Gaines, B.R., Schreiber, G., van Someren, M., Eds. **Current Trends in Knowledge Acquisition**. pp. 276-288. Amsterdam, IOS.

Piatetsky-Shapiro, G. & Frawley, W. (1991). **Knowledge Discovery in Databases**. Cambridge, Massachusetts: MIT Press.

Quinlan, J.R., Compton, P.J., Horn, K.A. & Lazarus, L. (1987) Inductive knowledge acquisition: a case study. Quinlan, J.R., Ed. **Applications of Expert Systems**. pp.157-173. Sydney: Addison-Wesley.

The Automated Knowledge Engineer: Natural Language Knowledge Acquisition for Expert Systems*

Yong Gao and Sharon Salveter
Department of Computer Science
Boston University
111 Cummington St.
Boston, MA 02215
gao@cs.bu.edu, salveter@cs.bu.edu

Abstract

This paper reports on the preliminary results of our research that addresses the problem of automating knowledge acquisition using natural language processing techniques. It describes the Automated Knowledge Engineer, a domain-independent interactive system that extracts a human expert's knowledge through natural language dialogue and encodes it into the form of frames and rules that are supported by many expert system shells. The important aspects of our research are 1) identification of the linguistic constructs used to define the concept structures and rules for expert systems, and 2) utilization of the dynamically-acquired knowledge base as an information source for natural language processing.

1. INTRODUCTION

We define *knowledge acquisition* (KA) as the process in which human expertise is elicited, organized and transferred into an expert system's knowledge base. Traditionally, KA is performed by human knowledge engineers who interview domain experts and manually encode the information in an appropriate formalism, a time-consuming and expensive process. Several approaches have been proposed and tools designed with the purpose of either automating or partially automating this process [Marcus, 1988]. However, these KA techniques, manual or automated, are still limited [Gruber, 1989]. KA remains the primary bottleneck in the efficient development of expert systems.

*This research has been partially supported by NSF grant IRI-8710137, the Digital Equipment Corporation and Tektronix, Inc.

This paper reports on the preliminary results of our research that addresses the *natural language knowledge acquisition problem*, namely, the problem of automating KA by applying natural language processing (NLP) techniques. It describes the Automated Knowledge Engineer (AKE), a prototype domain-independent knowledge acquisition system which extracts a human expert's knowledge through interactive natural language dialogue and encodes it into the form of frames and rules that are supported by many expert system shells such as KnowledgeCraft[Carnegie Group, Inc, 1985], PARMENIDES and FRulekit[Shell and Carbonell, 1990]. Our research focuses on KA for *classification expert systems*, a type of expert system that solves problems by identifying a set of objects based on their features [Boose and Gaines, 1989].

The purpose of the present research is two-fold: 1) to explore the solutions for some general problems of NLP by applying them in a particular task domain, and 2) to develop practical NLP techniques for building KA tools for expert systems.

1.1. Concept Structures and If-Then Rules as Focused, Organized Knowledge

Human experts' problem-solving knowledge can be categorized into domain knowledge and strategic knowledge [Clancey, 1983]. Domain knowledge can be classified as generic concepts that represent classes of objects, describing their attributes and values, and instance concepts that instantiate some generic concepts [Brachman and Schmolze, 1985]. The generic concepts and their overall hierarchical organization constitute the ontology of the domain. Strategic knowledge specifies the relevant legal operations on domain knowledge. Strategic knowledge can be expressed as strategic rules with the if-then format [Gruber, 1989].

We refer to the forms of knowledge described above as *focused, organized knowledge.* They correspond naturally to rules and frames, the most widely used knowledge representation formalisms in current expert system tools [Fikes and Kehler, 1985]. In the context of our research, we view the knowledge base of an expert system as a set of concept structures and if-then rules. A concept structure is a frame-like unit representing a concept in a knowledge base. It comprises the name of a concept, a set of attributes of the concept and a value restriction for each of the attributes. Concept structures can be organized as type hierarchies, using links such as "IS-A" and "HAS-PART." These structures are also used to define instance concepts, the elements in the working memory of an expert system. An if-then rule consists of an antecedent that specifies conditions to be matched against the elements in the working memory, and a consequent that specifies actions modifying the working memory. An inference engine is used to control execution of the rules. The diagram below displays the organization of frame-and-rule-based expert systems and the role of knowledge acquisition.[1]

[1] As a convention, rectangles are used for processes and ovals for data. In most cases, an arrow means data flow. In addition, arrows to or from rectangles also mean process control.

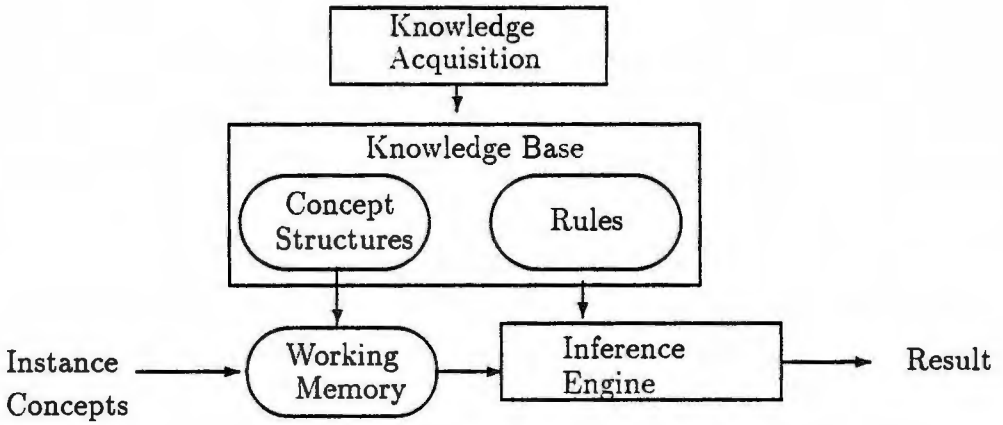


Fig. 1. Knowledge Acquisition and Expert Systems

The task of knowledge acquisition is only to encode the concept structures and rules into the expert system. The instance concepts are provided by the end users of the system as an instance problem of a domain. Without loosing generality, we use PARMENIDES and FRulekit as the core of IRL, the Internal Representation Language for the AKE system. Our task is to transfer the focused, organized knowledge, in the form of concept structures and if-then rules expressed in English, from human experts to the formalisms in IRL.

1.2. Automating the Knowledge Transfer through Natural Language Interaction

We propose to automate the transfer of the focused, organized knowledge using natural language processing techniques. Natural language is the major form in which humans express their knowledge. In particular, focused, organized knowledge can be articulated directly by human experts or documented by knowledge engineers who interview with them [Prerau, 1987] as natural language text.

The advantages of natural language over an internal representation language as a means of expressing knowledge are its learnability and expressiveness [Hayes, 1988]. The use of natural language allows domain experts to express their knowledge naturally and easily, allowing knowledge engineers to concentrate on more sophisticated aspects of system implementation.

The following examples illustrate how English utterances are used to express human expertise.[2]

1. A multiplexor is a device.
2. A multiplexor can be at a local or remote node.
3. A multiplexor has channels.

[2]These examples are part of the result of our analysis of the data in an electronic trouble-shooting expert system that we obtained from Tektronics, Inc.

4. The status of a channel can be normal or erroneous.

5. If the status of a channel of a multiplexor is not normal, and the node of the multiplexor is remote, then the channel is bad.

6. If the board contained in the multiplexor is bad,
   and the board is of a controller type,
   and the node of the multiplexor is local,
   then increase the channel-capacity of the multiplexor by 20 percent.

Utterances 1 through 4 are used to define concepts (or objects) in the ontology of the electronic trouble-shooting domain. Utterances 5 and 6 are English sentences for if-then rules.

As in other applications of NLP, specifying and representing the extensive information about the lexicon, syntax, semantics and specific domain are major obstacles to our success. In this paper, we present an approach that identifies the linguistic constructs used to define the concept structures in the domain ontology and rules for expert systems, and utilizes the dynamically-acquired knowledge base as an information source for natural language processing. Our goals are 1)to recognize the relationships between representations of knowledge at the linguistic level and conceptual level; 2) to process natural language in the context of KA by bridging the possible gaps between these two levels. The design of the knowledge acquisition system AKE is based on the above considerations.

Above we defined the goals and scope of our research and discussed the fundamental aspects of our approach to the natural language knowledge acquisition problem. In the rest of this article, Section 2 surveys the related research in this area . Section 3 describes the overall structure of the AKE system. Detailed analysis of the design of AKE and examples are provided in Sections 4 and 5. The final section summarizes our current research and outlines the direction of future work in this area.

2. RELATED RESEARCH

The use of natural language in automated knowledge acquisition may be traced back to the first knowledge acquisition tool TEIRESIAS [Davis and Lenat, 1982] for the medical diagnosis expert system MYCIN. TEIRESIAS had no parser; the English input of an expert's rule was processed primarily through keyword matching and by using a rule model. The weakness of the natural language processing techniques presented the major barrier to better performance of the TEIRESIAS system, as acknowledged by its designers, However, the idea of using a model of the task as a guiding structure for natural language processing plays an important role in our research.

NANOKLAUS [Hass and Hendrix, 1983] was designed with emphasis on acquiring knowledge for question-answering and fact-retrieval systems through natural language interaction with domain experts. The system was used to acquire new concepts from a user through

English dialogue. It raised, rather than dealt with, several important research problems of knowledge acquisition using natural language, such as the structuring of lexical entries so that new lexical information can be acquired and the identification of the linguistic constructions that are used to define new concepts. NANOKLAUS was the first system that attempted to acquire simultaneously both domain knowledge and the linguistic constructions used to express them. However, the system's capability to deal with natural language was restricted by its use of a pragmatic grammar.

Part of our research goal is similar to that of NANOKLAUS in that acquiring domain concepts, in addition to rules, is also our system's task. We differ from NANOKLAUS in that the acquired concept structures in the AKE system will serve in part as the semantic and domain knowledge for processing English rule definitions of an expert system.

UCTeacher [Martin, 1985], the acquisition component for UC, a natural language consultant program for the Unix operating system, was another system which provided for knowledge acquisition through a natural language dialogue with a domain expert. The acquisition process of the system depended on the predefined structures, or templates, of the domain knowledge. It would be extremely difficult to adapt it to other domains.

INKA [Phillips et al., 1985] was a natural language interface to a knowledge acquisition system in the electronic instrument troubleshooting domain. Its goal was close to ours in that it also dealt with knowledge acquisition for expert systems. In INKA, a rule definition in English was parsed and then translated into Prolog rules using a lexical-functional grammar. However, the system could only process sentences of simple forms such as *IF* ⟨*measurement results*⟩ {*WHEN* ⟨*setup procedure*⟩ *THEN* ⟨*diagnostic belief*⟩}. This not only limited the system to a very specific domain, but also seriously restricted the users' expressive flexibility. AKE will be much more powerful in its ability to deal with a broader range of English sentences because of the availability of the dynamically acquired domain concepts and grammar support.

INKA later evolved into a more powerful multi-paradigm knowledge acquisition tool ASTEK [Jacobson and Freiling, 1988], which allowed users to define knowledge through different media according to the nature of the knowledge. The natural language editor of the system was able to perform word prediction and word completion in order to improve input efficiency. The main advantage of ASTEK was its expressibility for knowledge forms other than natural language, such as tree-like structures for diagnostic procedures.

KBAM [Silvestro, 1987] was a knowledge acquisition system based on the identification of explanatory utterance types, such as definitional, causal and concept-relation. While it is not clear why these types of utterances are necessary or sufficient to express and organize knowledge in general, this approach is appropriate for specific tasks or domains. We adopt the idea of utterance type to classify the linguistic constructs for expressing English definitions of concepts and rules.

More recently, the SNOWY system [Gomez, 1989] was developed to automatically acquire concepts and build the classification hierarchies for a consultant expert system from natural language texts. A paragraph of English text was first parsed into logical forms from which conceptual structures were built by using formation rules. Then these concepts were

recognized and integrated into type hierarchies. In this aspect, SNOWY is similar to the concept-defining component of our AKE system. However, because AKE is to be used to acquire knowledge for rule-based expert systems interactively from users, not from arbitrary English text, the forms of linguistic constructs that may be used to define concepts are restricted by the underlying knowledge representation formalisms of an expert system shell. This allows us to identify the linguistic constructs used to express concepts in the context of expert systems shells. Moreover, the SNOWY system can only be used to construct classification hierarchies, while our system is used to acquire both concepts and rules.

3. AKE: THE AUTOMATED KNOWLEDGE ENGINEER

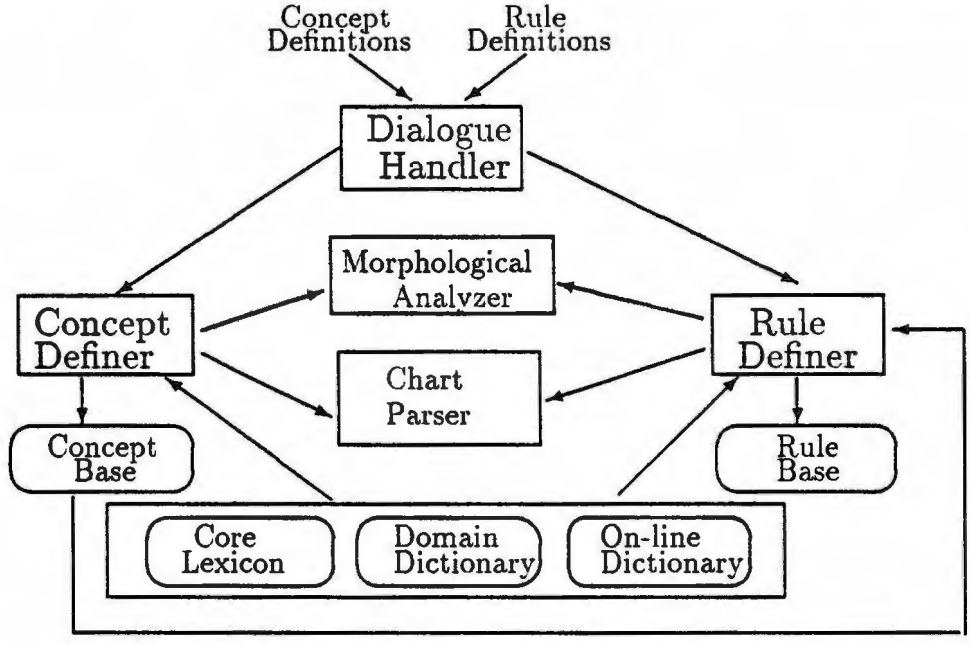


Fig2. The Organization of AKE

This section outlines the design of the AKE system, which can be used to assist human experts or knowledge engineers in defining concept structures and if-then rules in English and transform them into IRL formalisms. Figure 2 shows the overall organization of AKE.

The main components of the AKE system are a concept-defining module, the Concept Definer, and a rule-defining module, the Rule Definer. The Dialogue Handler interacts with the user and decides which module to invoke. English definitions of concepts or rules are processed by a morphological analyzer and a syntactic parser, common to both modules. However, different sets of morphological and syntactic rules are used in different modules. The output of the syntactic parser is translated into internal representations by a construction phase. The core of the syntactic component is an active chart parser [Winograd, 1983] that makes use of a collection of rewrite rules. Also common to both modules is a domain-independent core lexicon that contains primarily words of closed classes, such as prepositions, key words and auxiliary words as explained in the next section. When an unknown word

occurs, the machine-readable version of Webster's Seventh Collegiate dictionary is consulted. Although the system can be used in concept-defining mode or rule-defining mode, we assume that the concept structures are defined before a rule can refer to them. This is because, as indicated in [Bennett, 1985], the identification of the conceptual structures is an important prerequisite of the expression of rules. However, this restriction can easily be relaxed. Whenever a rule refers to a concept structure that does not exist in the current knowledge base, the system automatically changes its mode to define a concept.

The ultimate result of processing an input is a piece of information that augments the developing concept/rule base. This base is itself a knowledge source to the natural language processing component and represents domain-dependent knowledge. As the concept/rule base becomes larger, the accuracy of the parsing process will increase, requiring less interaction with the user to resolve ambiguity.

In the rest of this section, we discuss some fundamental considerations on which the system is based. The next two sections will emphasize on the design of the Concept Definer and Rule Definer in detail.

3.1. Identifying the Linguistic Constructs

In general, NLP is difficult because of the inherent ambiguities and complexities of natural language. Attempts to design systems covering broad language phenomena have yet to be successful. We believe that the success of a practical NLP system relies on the analysis of the characteristics of the specific task the system is designed for. In particular, the identification and classification of the linguistic constructs that can be used in the task enables a NLP system to deal with natural language in a controlled domain without sacrificing its users' expressibility. Furthermore, this allows us to take different approaches to different types of linguistic constructs.

As indicated in the first section, knowledge for a rule-based expert system can be classified into concept structures and if-then rules. The type and complexity of the natural language descriptions for these two forms of knowledge are quite different. More specifically, we have observed that, simple sentences, mainly sentences with "be" and "have" as the main verb, such as "a multiplexor is a device" and "multiplexers have channels" are often sufficient to describe the information about a concept structure, namely the name of a concept, its attributes and their values, the taxonomic relations between two concepts. For example, since the subject, or the focus of a concept-defining sentence can only be a concept name or an attribute name of a concept, its syntactic patterns are very limited. When the subject is a complex noun phrase, the sentence is used to define a rule, not a concept structure. For example, the sentence "A creature with wings and feathers is a bird" is actually equivalent to "If a creature has wings and feathers, then the creature is a bird." It is different from "A bird is a creature with wings and feathers", a concept-defining sentence.

Based on this analysis, several utterance types similar to those of KBAM are identified, and a set of construction rules are used to map syntactic structures of these utterance types into actions creating or modifying concept structures, as will be discussed in Section 4.

3.2. Utilizing the Already-Acquired Knowledge

On the other hand, the English definitions of if-then rules are often much more complex in their structures, because, for example, a condition of an antecedent may test several concepts and the antecedent of a rule may involve arbitrary boolean combinations of the tests, as shown in the examples given in Section 1. To deal with sentences of this complexity in a non-trivial domain-independent NLP system, one needs an extensive lexicon and knowledge of syntax, semantics and pragmatics. A traditional approach is to separate the domain-independent knowledge, such as syntax, from that specific to a domain. An acquisition module must be designed to acquire domain knowledge, as in many natural language interfaces to databases [Ballard et al., 1986; Grosz et al., 1987]. This puts an extra burden on the designers of these interfaces. Our solution to this problem is to take advantage of a particular feature of our task of acquiring focused, organized knowledge for rule-based expert systems. That is, the already-acquired dynamically-changing knowledge base being built is available as an information source for processing new English definitions of knowledge. The output of AKE is a knowledge base for both the expert systems and AKE itself. In particular, the defined concept structures can be used as guiding structures for processing a rule definition. Because, particularly in classification expert systems, a rule can only refer to the content of concept structures already defined, ambiguity is significantly reduced. Moreover, as we will explain in Section 4.1, the existing concept structures can also be used to help interpret a new concept structure definition.

In summary, the natural language definitions of concept structures are processed first without requiring much pre-defined information. Then the acquired concept structures are used to help process the much more complex if-then rule definitions.

The idea of using the knowledge base as an aid to NLP was independently suggested by [Hayes, 1988] to design interfaces to expert systems. A similar approach was also taken by [Harris, 1977] to process natural language queries to databases. Our research represents a systematic treatment of this idea in knowledge transfer for rule-based expert systems. Moreover, our approach is based on the recognition of the necessary linguistic constructs used to define ontological concept structures and rules for expert systems, and features the utilization of the concept structures as a guiding role in translating rules into their internal representations.

4. THE CONCEPT DEFINER

The task of the Concept Definer is to construct the ontology of a problem domain from the English descriptions of concepts, their attributes and possible values of these attribute, as well as the relationships among them. Based on the analysis of the linguistic constructions that can be used for these descriptions, the Concept Definer is designed such that an English sentence is processed through morphological analysis, word categorization, syntactic parsing and final concept construction.

Given an English description of a concept or concepts, the first step of the Concept Definer is to characterize the words in the sentence into the following groups:

1. Content words, such as "multiplexor" and "channel," are the actual terms stored in concept structures.
2. Key words, such as "concept" and "attribute," are terms used in the IRL.
3. Functional words, such as prepositions, articles, and quantifiers, play specific syntactic functions in a sentence.
4. Auxiliary words, such as "contain" and "kind," have a special meaning in the context of our task.

Content words are placed in concept structures as the names for concepts, attributes and values. Words of other groups are used to help the Concept Definer to locate the right positions for the content words.

At the same time, new words, mostly the content words, are defined with the help of the morphological analyzer and an online dictionary.

Then, the parser produces a syntactic logical structure of the input, that is in turn processed by the Concept Construction phase, as shown in the following figure.

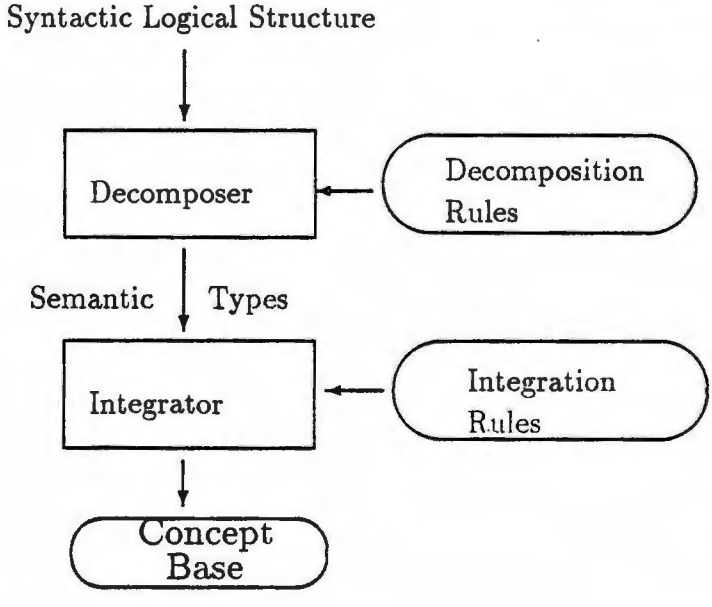


Fig. 3. The Concept Constructor

At the Decomposition step, the decomposition rules associated with the nodes of the syntactic logical structure are triggered , mapping these structure into one or several semantic types, each of which corresponds to a CREATE or MODIFY function in IRL. Currently, the following four semantic types are identified:

1. Concept-name-description type.
   For example, in "a device is a CONCEPT," the keyword CONCEPT and the IS-A pattern indicates that DEVICE is a concept name. The CREATE function is applied to DEVICE.

2. Taxonomy-and-relation description type.
   This type is used to explicitly describe relations between concepts. For example, given "a multiplexor is a device," an IS-A link will be created between MULTIPLEXOR and DEVICE, using the MODIFY function.

3. Attribute-description type.
   For example, given "a multiplexor has a channel-capacity," CHANNEL-CAPACITY will be added as an attribute to the concept structure for MULTIPLEXOR, using the MODIFY function.

4. Attribute-value-restriction type.
   For example, in "the capacity of a channel can be low or high" value-restriction LOW and HIGH will be added to the attribute CHANNEL-CAPACITY. [3]

The following is an example of decomposition rules associated with noun phrases:

> *If* the noun phrase has the pattern "ADJECTIVE1 or ADJECTIVE2 NOUN."
> *then* define concept for NOUN if it is new, *and* add ADJECTIVE1 *and*
> ADJECTIVE2 as possible values of the attribute UNKNOWN for NOUN.

At the next Integration step, the semantic types created by the Decomposition step are combined and integrated into consistent and unambiguous concept structures by the integration rules. For example, the following rule tries to use the information defined previously to put the possible values in the UNKNOWN attribute into their proper positions:

> *If* V1 *and* V2 are possible values of the UNKNOWN attribute of concept C;
> *and* SUPC is a super concept of V1 *and* V2,
> *and* SUPC is not an attribute of C,
> *then* define SUPC as an attribute of C,
> *and* add V1 *and* V2 as possible values of this attribute.
> *else if* one of V1 *and* V2 is a possible value of the attribute ATTR of C,
> *then* add the other one as a possible value of ATTR of C.

This rule may be applied to the noun phrase "local or remote node." If the knowledge base already knows that "local" and "remote" are "locations," then a new attribute "location" is created for the concept "node" and both "local" and "remote" are added as possible values of this attribute. This rule shows the use of previously-defined concept ("location") to define new concept ("node").

[3] Discourse analysis techniques should be used here to ensure that CHANNEL is not a concept name and CHANNEL-CAPACITY is an attribute of concept MULTIPLEXOR.

5. THE RULE DEFINER

The organization of the Rule Definer is similar to that of the Concept Definer, as shown in Fig. 4.

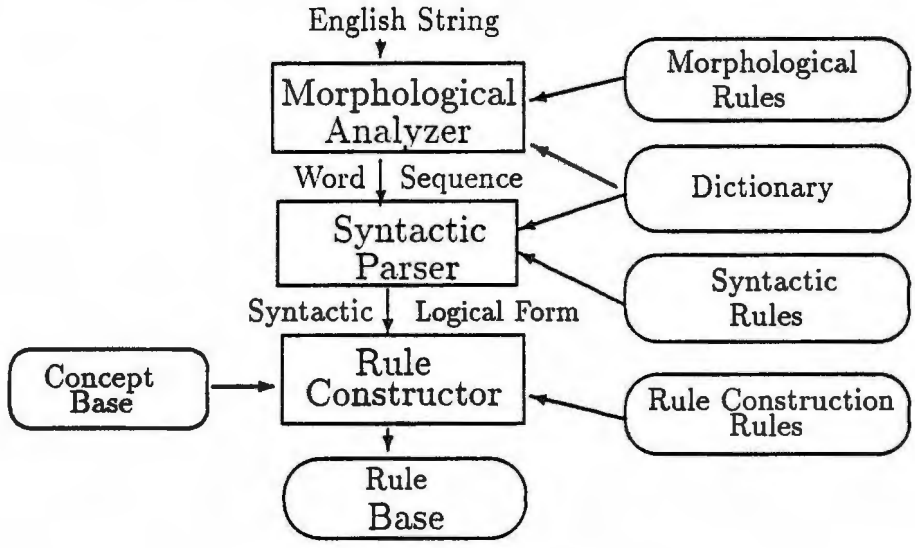


Fig 4. The Rule Definer

The syntactic analysis phase of the Rule Definer parses the rule description in English into a tree-like syntactic logical structure. This structure is then interpreted and encoded into a rule representation in IRL format at the rule-construction phase. Traditionally, the second phase of this process is called semantic interpretation, which is performed by using the semantic information about the lexical items that appear in the sentence. Because of our domain-independence requirement, this kind of information cannot be pre-defined in AKE. Thus the traditional semantic interpretation methods cannot be used directly in the rule-construction phase. Our approach is to use the concept structures already defined by the Concept Definer as semantic information to guide the rule-construction process.

Although the Rule Definer has separate components for syntactic and semantic knowledge, in order to achieve modularity [Garfield, 1987], the entire process of translating the rules is incremental [Mellish, 1985]. This allows the parser to call the semantic interpreter when it is needed, and the interpreter starts to process part of the unfinished sentence whenever a condition, or the antecedent is parsed. This architecture also increases the power and efficiency of the parser.

5.1. Concept-Structure-Driven Rule Construction

Structure-driven processing is a well-known method in psychology which has been used to explain some cognitive process, for instance, vision [Davis and Lenat, 1982, p. 304]. It is also

a technique widely used in natural language understanding. The main idea of this approach is that, in order to process a segment of language input, such as a phrase, a sentence or a paragraph, a pre-stored structure such as a case-frame [Bruce, 1975] or script [Schank, 1977] is selected and the ensuing processing of the input is guided by the structure in a slot-filling fashion. Usually, these guiding structures must be pre-defined and thus domain-dependent. Our contribution to this approach is employing the already-acquired concepts as the guiding structures.

In the context of rule-based expert systems, an if-then rule represents test(s) on a state of the expert system and action(s) on the state, which are based on the test. The state is represented in terms of the contents of instantiated concept structures, or elements in the data memory of the expert system. In other words, the tests and actions of a rule refer only to the defined concepts and their attributes and values. Thus, as we indicated earlier in this paper, in an English description of a rule, the information referred to is only that defined in the concept structures.

A skeleton of the algorithm of our rule constructor is as follows:

1. Separate the input sentence into antecedent and consequent parts.
2. For each condition clause in the antecedent or action clause in the consequent, find the concept name.
3. The remaining constituents of a clause are matched against the concept structure associated with the concept name in a top-down fashion. A triplet with the form <concept-name><attribute-name><value-name> is created.
4. For each of the actions, in addition to the triplet, find the action type (modify, delete, ...) and create the action representation.

As an example, we describe how the Rule Definer interprets the following rule definition sentence:

> If the board contained in the multiplexor is bad,
> and the board is of a controller type,
> and the node of the multiplexor is local,
> then increase the channel-capacity of the multiplexor by 20 percent.

We assume the user has defined the following concept structures expressed here in IRL formalism:

```
(CONCEPT-NAME node
 (ATTRIBUTE location
  (POSSIBLE-VALUES: local, remote)))

(CONCEPT-NAME multiplexor
 (IS-A device)
```

```
(ATTRIBUTE node
  (POSSIBLE-VALUES: node)))
(ATTRIBUTE parts
  (POSSIBLE-VALUES: board, selector)))

(CONCEPT-NAME board
 (IS-A device)
 (ATTRIBUTE type
  (POSSIBLE-VALUES: controller, channel-interface))
 (ATTRIBUTE status
  (POSSIBLE-VALUES: good, bad, ok)))
```

The Rule Definer processes a sentence clause by clause, mapping each clause into one or more IRL condition tests or actions. For example, for the clause "board contained in the multiplexor is bad," the parser produces two subtrees corresponding to "multiplexor contains board" (notice that the voice is changed) and "board is bad." "Contain" is an auxiliary word that the domain-independent dictionary indicates as synonymous to "has as part." By their positions in the subtrees, "multiplexor" and "board" are taken as names for the concept structures, whose contents are then used to guide the processing of the subtrees. A subtree having triple nodes, such as "multiplexor contains board," is mapped into a condition test in IRL directly. On other hand, in "board is bad," "is" is a functional word which is not an attribute in the concept structure for "board." Therefore, each attribute of "board" is checked to see if "bad" can be its possible value. In this case, "status" is found. We call "status" an *implicit attribute* in the phrase "board is bad." The rest of the sentence is processed in a similar manner.
The final representation in IRL of this rule is given as follows.

```
((and ($multiplexor part $board)
      ($board status bad)
      ($board type controller)
      ($node location local))
-->
 (modify ($multiplexor channel-capacity +20%)))
```

A $ sign indicates a variable that can be matched to an instance concept in the data memory.

5.2. Knowledge Representation Gaps

The organization of the concept structures, their attributes and values for an expert system does not necessarily suit the purpose of natural language processing. There exist gaps between the representation of knowledge on conceptual level and linguistic level. Currently, two types of representation gaps between the user-defined concept structures and those required by semantic interpretation are recognized:

1. The minimum meaningful unit problem.
   Some composite terms, like "first-time-home-buyer", are often *minimum meaningful*

units in the knowledge base of an expert system. They may not need to be represented as several individual concepts. However, the words comprising the term may be used individually in sentences to refer to this concept. The linguistic information must be available to process these sentences.

2. Verb-centered semantics and object-centered concept organization.
   For example, sentences such as "A professor teaches courses" is usually processed with the help of a verb-based case-frame with "teach" as the head and "agent" and "object" as cases. While the same information in an expert system can also be organized either around the noun "professor" or "course."

A systematic treatment of these knowledge representation gaps has yet to be formalized. One solution is to use a separate set of semantic case-frames to process the sentence and then map the result into rules. This is not appropriate to our purpose because the case-frames are not available. We plan to use the concept structures directly and provide some mapping rules to bridge the gaps if they exist.

Methods of dealing with other difficult problems also need to be provided. These problems include the possible ambiguity caused by the situation where a value may belong to several attributes, or an attribute may belong to several concept names, and the nested structure problem, where a simple clause may involve several concept structures.

6. CONCLUSION AND DISCUSSION

Allowing a human expert to express knowledge through natural language is a desirable feature of a knowledge acquisition system. By identifying the necessary linguistic constructs used to define ontological concept structures and rules for expert systems, and taking advantage of the availability of the concept structures and if-then rules being defined as a knowledge source, it is possible to make this difficult task feasible. In this paper, we have shown how a dynamically-acquired knowledge base can be used to support automated transfer of organized knowledge for rule-based expert systems through natural language interaction. Our approach is to first process the English definitions of ontological concept structures based on their utterance types. Then these user-defined concepts are used as guiding structures for interpreting more complex English descriptions of if-then rules. To achieve domain-independence, a morphological analyzer is used and an online dictionary is consulted. We have described the design of a prototype natural language knowledge transfer system AKE based on this approach. Our research focuses on extracting the organized knowledge from a human expert through natural language interaction and transforming it into the form of frames and rules that are supported by many expert system shells.

By using natural language techniques, our system allows human experts to express their knowledge in linguistic forms directly. Because our approach does not assume any initial domain-dependent information, but learns it as a result of natural language processing, we expect our results to be applicable to a wide variety of tasks where natural language front ends are desirable, especially natural language interfaces for expert systems.

6.1. Status of Current Research and Future Work

This paper described only the fundamental assumptions and principal approaches of the present research. The focus of our future research will be the acquisition of lexical information from the utterances given by the user and the utilization of the online dictionary as an information source to help the human expert define their domain concepts.

The AKE system is being implemented in Austin Kyoto Common Lisp on a Sun 4/280 machine. We expect to test the system in two domains: troubleshooting for electronic devices and for automobiles. The evaluation of the system will be conducted both by our analysis of the output (concepts and rules) of a knowledge transfer session using AKE, and by comparing the output of AKE with the expert system produced by a knowledge engineer for the same problem domain.

References

[1] Ballard B. W. and Stumberger, D. E. (1986). Semantic Acquisition in TELI: A Transportable, User-Customized Natural Language Processor. In *Proc. of the 24th Ann. Mtg. of the ACL*, New York.

[2] Bennett, J. S. (1985). ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. *Journal of Automated Reasoning*, **1**:49-74.

[3] Boose, J. H. and Gaines, B. R. (1989). Knowledge Acquisition for Knowledge-Based Systems: Notes on the State-of-the-Art. *Machine Learning*, **4**: 377-394.

[4] Brachman, R. J. and Schmolze, J. G. (1985). An overview of the KL-ONE Knowledge Representation System. *Cognitive Science* 9(2):171-216.

[5] Bruce, B. (1975). Case Systems for Natural Language. *Artificial Intelligence*, 6(4):327-360.

[6] Carnegie Group, Inc. (1987). *Knowledge Craft V 3.1.*

[7] Clancey, W.J. (1983). The Epistemology of Rule-based Expert System: A Framework for Explanation. *Artificial Intelligence*, **20**:215-251.

[8] Davis, R. and Lenat, D. B. (1982). *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill International Book Company, New York.

[9] Fikes, R. and Kehler, T. (1985). The Role of Frame-Based Representation in Reasoning. *Communications of the ACM*, **28**(9):904-920.

[10] Garfield, J. L. (Ed.), (1987). *Modularity in Knowledge Representation and Natural Language Understanding*, Cambridge, Mass: MIT Press.

[11] Gomez, F. and Segami, C. (1990). Knowledge Acquisition from Natural Language for Expert Systems Based on Classification Problem-Solving Methods. *Knowledge Acquisition* **2**: 107-128

[12] Grosz, B. J., Appelt, D. E., Martin, P. A. and Periera, F. C. N. (1987). TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces. *AI*, **32** (2), 173-243.

[13] Gruber, T. R. (1989). Automated Knowledge Acquisition for Strategic Knowledge. *Machine Learning*, **4**: 293-336.

[14] Haas, N. and Hendrix, G. G. (1983). Learning by Being Told: Acquiring Knowledge for Information Management. In Michalski, R. S., Carbonell, J. and Mitchell, J. M. (Eds.), *Machine Learning: An AI Approach*, Palo Alto, CA, Tiog Press, 405-427.

[15] Harris, L.(1977). *Using the Database Itself as a Semantic Component to Aid the Parsing of Natural Language Database Queries.* Dartmouth College Mathematics Department TR 77-2.

[16] Hayes, P. J. (1988). Using a Knowledge Base to Drive an Expert System Interface with a Natural Language Component. In J. Hendler (Ed.), *Expert Systems: the User Interface.* Ablex Pub. Corp., Norwood, New Jersey.

[17] Jacobson, C. and Freiling, M. J. (1988). Astek: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge. *Int. J. Man-Machine Studies*, **29**:311-327.

[18] Marcus, S. (Ed.), (1988). *Automating Knowledge Acquisition for Expert Systems.* Boston: Kluwer Academic Publishers.

[19] Martin, J. H. (1985). Knowledge Acquisition Through Natural Language Dialogue. *2nd Conf. on AI Application*, 582-586.

[20] Mellish, C. S.(1985). *Computer Interpretation of Natural Language Descriptions.* Ellis Horwood Limited, Chichester, England.

[21] Prerau, D. (1987). Knowledge Acquisition in the Development of a Large Expert System. In *AI Magazine* **8**(2):43-51.

[22] Phillips, B, Messick, S. L, Freiling, M. J. and Alexander, J. H.(1985). INKA: The INGLISH Knowledge Acquisition Interface for Electronic Instrument Troubleshooting Systems. *2nd Conf. on AI Application*, 676-681.

[23] Schank, R.C. and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*, Hillsdale, NJ: Lawrence Erlbaum.

[24] Shell, P. and Carbonell, J. (1990). *FRulekit: A Frame-Based Production System - User's Manual.* Carnegie Mellon University, Version 1.5, Feb., 1990.

[25] Shell, P. and Carbonell, J. (1990). *PARMENIDES: A Class-Based Frame System - User's Manual.* Carnegie Mellon University, Version 1.5, Feb., 1990.

[26] Silvestro, K. (1988). Using Explanations for Knowledge-base Acquisition. *Int. J. Man-Machine Studies* **29**: 159-169.

[27] Winograd, T. (1983). *Language as a Cognitive Process: Syteax.* Addison-Wesley: Reading, MA.

The Acquisition of Word Syntactic Knowledge from Sample Sentences

Fernando Gomez
Department of Computer Science
University of Central Florida
Orlando, Fl 32751
gomez@cs.ucf.edu

Abstract

Learning algorithms which acquire syntactic knowledge for a parser from sample sentences entered by users who have no knowledge of the parser or English syntax are described. It is shown how the syntax of words can be inferred from sample sentences entered by end-users. Then, if the parser fails to parse the sentence, say "Peter knows how to read books," because it has limited knowledge or no knowledge at all about "know," an interface, which incorporates the acquisition algorithms, can be activated and "know" be defined by entering some sample sentences, one of which can be the one which the parser failed to parse. The interface to the parser is one of the main components of SNOWY, a system which acquires knowledge from texts.

1 Introduction

One of the stumbling blocks of systems designed to acquire knowledge from text is that of syntax acquisition. If the parser used by the knowledge acquisition system is unable to parse a user's sentence, then the acquisition process ends abruptly. Formal knowledge about syntax is a high form of expertise knowledge which most people do not have. Moreover, to transfer this knowledge to the programs which embody the syntactic knowledge is not an easy matter either. The ideal situation would be one in which the end-users would communicate the syntactic knowledge to the knowledge acquisition system by typing some sample sentences. This is the approach that we explore in this paper. We describe learning algorithms which acquire syntactic knowledge for a parser from sample sentences entered by users who have no knowledge of English syntax or the parser. We will refer to these learning algorithms as the interface to the parser, or simply the interface. This interface is one of the main components of SNOWY [Gomez and Segami, 1989; Gomez and Segami, 1991]. The methods used to acquire world knowledge are described in [Gomez and Segami,

*This research is being funded by NASA-KSC Contract NAG-10-0058

1991]. In that paper, it was indicated that the automatic acquisition of syntactic knowledge was a necessary element of any system designed to acquire knowledge from texts, and that an interface which would allow end-users to increase the capabilities of the parser was under construction. This paper describes that interface.

Our parser is a lexically driven algorithm which incorporates mechanisms to handle "general" syntactic phenomena such as long distance dependency, embedded clauses, passive sentences, syntactic ambiguity, etc. However, the enormous syntactic diversity of most words in English is handled by storing transition diagrams in the lexical entries of the words. The parsing of a sentence is driven by these syntactic usages, which are retrieved by the parser as they are encountered in the sentence. Thus, a word like "news" may be followed by a clause introduced by the conjunction "that." Or a word like "way" may be followed by an infinitive clause. This behavior is idiosyncratic of "news" and "way." Likewise, the verb "know" may be followed by "how," but "want" followed by "how" will result in an ungrammatical sentence. The approach described in this paper has been inspired by our experience in learning a second language. How does a non-native speaker of a language encode his/her knowledge about a second language? Our answer is that learning a language hinges on mastering the *syntactic usages* of individual words rather than on learning general syntactic rules. In this paper, we will make clear that one of the advantages of our parsing algorithm is that it facilitates the acquisition of syntactic rules from naive users. Formalizing the syntax of words like "know," "be," etc. in a transition diagram is not easy for people with no knowledge of linguistics. In fact, our experience has shown that even college graduates have a lot of trouble in capturing the syntax of words in a transition diagram. In this paper, we show how the syntax of words can be inferred from sample sentences entered by end-users. Then, if the parser fails to parse the sentence, say "Peter knows how to read books," because it has limited knowledge or no knowledge at all about "know," the interface can be activated and "know" defined by entering some sample sentences. The appendix contains a sample session indicating how this interaction may proceed. The reader may want to read that sample session prior to delving into the paper.

A recent effort with a similar aim as this, but based on elicitation techniques is [Carter, 1989]. Work on transportable natural language interfaces [Ballard and Tinkham, 1984; Grosz et al. 1987] bears only an indirect relation to our work, since those authors designed their systems as front ends for data base systems.

This paper is organized as follows: Section 2 gives a very brief overview of the parser. Sections 3 to 6 describe the algorithm and the acquisition of usages for verbs, adjectives and nouns. Section 7 explains the problems with ambiguity, and section 8 gives our conclusions.

2 Overview of the Parser

This section describes briefly those features of the parser which provide the necessary context to understand the interface acquisition algorithms. An early version of the parser may be found in [Gomez, 1988], and a later version with some important revisions in [Gomez, 1989]. The parser is a top down algorithm which uses a "grammar" indicating the

structure of the sentence *up to the main verb.* There are two main stacks in the parser. One stack, called *non-terminals*, contains the non-terminal symbols, and a stack, called *actions*, contains the actions to be executed when a non-terminal is reduced from the *non-terminals* stack. The algorithm consists of two steps. In the first step, the *non-terminals* stack is initialized with one rule of the "grammar" and the *actions* stack is initialized with the actions for those non-terminals. In the second step, the symbols in the stacks are manipulated. By way of example, consider the following very simple "grammar" (cf. Gomez, 1988).

```
S --> SUBJ VERB
    | INF VERB
    | INF SUBJ VERB
```

The first rule of the "grammar" says that an English sentence consists of a subject followed by the inflected form of a verb. (The non-terminal VERB refers always to the inflected form of the verb). Whatever follows the verb depends on each concrete verb and, consequently, is not part of the "grammar". When the non-terminal VERB is reduced, its syntactical "usage" will be retrieved from the parser's lexicon entry and pushed on the stack. Similarly, the nouns in the subject may have special usages, in which case those usages will be pushed on the stack during execution of the NP (noun phrase) procedure. The second production of the "grammar" says that an English sentence is formed by an infinitive followed by a verb (to read is good), while the third production says that an English sentence is formed by an infinitive followed by a subject and a verb. In this case, the infinitive introduces a purpose clause as in "to read Mary went to the library."
The syntactic usage of a word is represented in the lexicon by a list of the form (a1 a2 ... ai), meaning that the word may be followed by just one of the ai elements. If an asterisk (*) belongs to the list, it means that none or one ai may follow the word. If any ai is a list, it means that the word may be followed by the first element in ai. This first element in turn may be followed by any number of elements in ai. The usage of "eat" is represented by (obj *) which means that "eat" may or may not take an object. Consider the usage of "want:"

```
want obj
want inf
want subj inf
```

This usage is represented as (inf (obj (inf *))), which corresponds to the structure (a1 a2), where a1 matches inf and a2 the list (obj (inf *)). The meaning of this representation is that "want" may be followed by an infinitive or object, and the object in turn may be followed by an infinitive or nothing. A transition diagram representing the usage of "want" and "like" is depicted in Figure 1. The actions corresponding to this usage are:

```
(same-subj (obj (diff-subj *)))
```

There is a one-to-one mapping between the symbols in the usage and the symbols in the actions. The symbols in the actions are names of functions which will be executed when

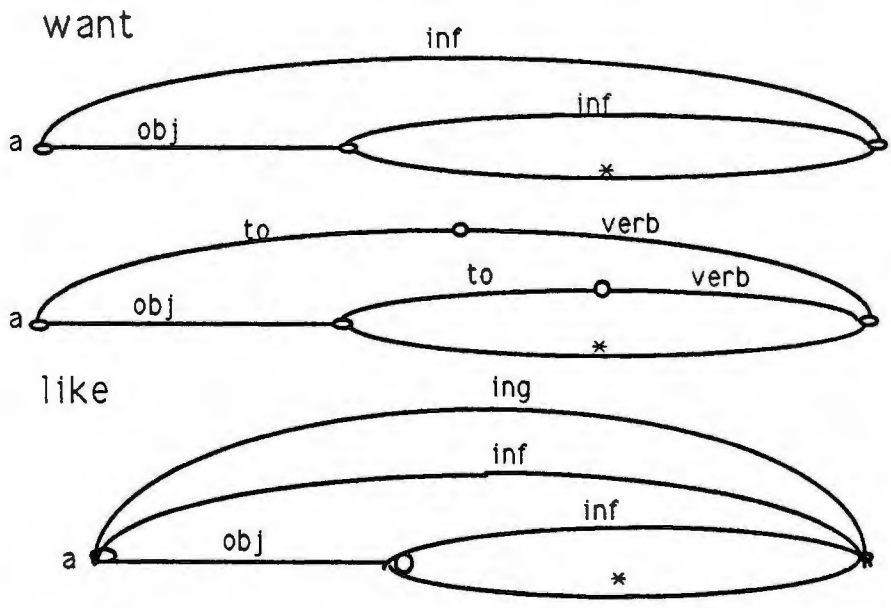


Figure 1: *Transition network representing the usages of "want" and "like." For "want," two possible representations are depicted.*

their matching symbols in the usage are reduced. Note that the representation of the usage of "want" assumes that the NP following the verb is a direct object. However, this assumption will be corrected by the action *diff-subj* if the NP is followed by an infinitive. This action will make the NP the subject of the embedded clause introduced by the infinitive. An alternative representation of the usage of "want" is shown in Figure 1. In that case, the action *same-subj* would be associated with the first instance of "to" and the action *diff-subj* with the second occurrence of "to." These actions could take care of the verb following "to." However, we have opted for taking care of infinitives in the lexical analyzer function called *advance*. This function takes care of the entire verb sequence.

The key idea in the algorithm is the function *coalesce* which selects a non-terminal from the *non-terminal* stack on the basis of the senses of the word being scanned. The selection of the correct arc or non-terminal is determined by matching the non-terminal or non-terminals on the top of the *non-terminals* stack to the senses in *wordsenses*. For instance, *coalesce* selects an arc labeled *inf*, if "inf" belongs to *wordsenses*; an arc labeled *verb*, if "verb" belongs to *wordsenses*; an arc labeled *that*, if the current word is "that," etc. We use the feature *descr* to mark all those words which can form part of a noun group. Thus, adjectives, nouns, articles, pronouns and the -ing form of verbs are marked with the feature *descr*. Hence, the function *coalesce* selects a non-terminal labeled *subj*, or *pred* (predicate) if *descr* belongs to the senses in *wordsenses*. *Coalesce* selects the non-terminal *obj* or *io* (indirect object) if *descr* belongs to *wordsenses* and the marker *nominative* does not belong to it. If *coalesce* selects a non-terminal in this way, we say that a *match* between the non-terminal and the senses in *wordsenses* takes place.

If the top of the *non-terminals* stack contains a single non-terminal, and a *match* exists between that terminal and the senses in *wordsenses*, *coalesce* returns true; otherwise it returns false. If *coalesce* returns true, the top of the *actions* stack is executed, which is the action corresponding to the non-terminal on top of the *non-terminals* stack. Every action, when it is executed, will pop both stacks and, if appropriate, will push a usage and a corresponding action on the *non-terminals* and *actions* stacks, respectively.

Let us now explain the behavior of *coalesce* when the top of the stack is a list. Refer to Figure 1. We call the node marked *a* the *initial* node and the arcs coming from the *initial* node *initial* arcs. The *initial* arcs for the usage of "want" are *inf* and *obj*. Suppose that the top of the stack contains (a1 a2), where a1 is a single non-terminal and a2 is a list. The function *coalesce* will select between a1 and the first element in a2. In other words, what *coalesce* does is to coalesce the *initial* arcs for the usage of a word. If the usage of "want" is being considered, *coalesce* will try to select the arc labeled *obj* or the arc labeled *inf*. If *coalesce* selects one arc, it will discard the remaining *initial* arcs. It may happen that in some cases *coalesce* will fail to coalesce the *initial* arcs. In that case, the saving of some arcs is needed (Gomez, 1989).

For example, consider the sentence "Peter wanted Mary to read a book." When the non-terminal *verb* is reduced, the usage and actions for "want" will be pushed on the *non-terminals* and *actions* stacks, respectively. Then *coalesce* will be activated with (inf (obj (inf *))) on top of the *non-terminals* stack and "Mary" as the current word. *Coalesce* will return true, pop the stack and push "obj (inf *)" on it. *Coalesce* will perform a similar change in the *actions* stack. Namely, it will pop it and push "obj (diff-subj *)" on it, so that the correspondence between non-terminals and actions is kept.

Relative clauses and prepositional phrases are handled in the following way. When the function that parses noun groups is ready to exit, it pushes (prep np pp rel ing-rel *) onto the *non-terminals* stack, and (preposition rel-np part-rel relatives ing-rel-act *) onto the *actions* stack. The non-terminal *prep* stands for preposition, *rel* stands for "which," "who" and "whom." The non-terminal *pp* stands for past-participle. Then, it is up to *coalesce* to select the correct arc. *Coalesce* will select the non-terminal *prep* if the marker *prep* belongs to the senses in *wordsenses*; *ing-rel* if the marker *ing* belongs to *wordsenses*; "that" if the the current word is "that;" *rel* if *rel* belongs to the senses in *wordsenses*; *np* if the marker *descr* belongs to *wordsenses*; or *pp* if the marker *pp* belongs to *wordsenses*. The selection of any of these non-terminals is followed by the selection of its corresponding action.

The *non-terminals* stack holds two special symbols: $ and +. The dollar sign indicates that the stack is empty. The + sign is used to indicate the end of the non-terminals for a sentence or an embedded clause. When the parser finds the + sign, it knows that a sentence or clause has been parsed, and therefore, that the structure which it is building for that sentence or clause is complete. The + sign is put on the stack by the initialization step and whenever an embedded clause is found by the action corresponding to a non-terminal.

The following detailed example will give the reader a feeling of how the parser works. Consider the sentence "The book Peter donated vanished."

```
(1) (subj verb + $)->(the book peter donated vanished)
(2) ((prep np pp rel ing-rel *) verb + $)->(peter ...)
```

```
(3) (subj verb + verb + $)->(peter donated vanished)
(4) ((prep np pp rel ing-rel *) verb + verb + $)->(donated ...)
(5) (verb + verb + $)->(donated vanished)
(6) (obj (io for to *) + verb + $)->(vanished)
(7) (io for to *) + verb + $)->(vanished)
(8) (+ verb + $)->(vanished)
(9) (verb + $)->(vanished)
(10) (+ $)->nil
(11) ($)->nil
```

When we refer to the stack, we will always mean the *non-terminals* stack. The initialization step has initialized the non-terminals stack to "subj verb + $" and the *actions* stack (not shown in the example) with the actions corresponding to those non-terminals. Also a stack called *structs* contains the name of the structure (a gensym) being built for the parse of the sentence. So *structs* is equal to (g01). Line 1: Coalesce matches *subj* to the senses of the current word, "the," and returns true. The action corresponding to the non-terminal *subj* calls the noungroup function which returns "the book." Also the noun group function inserts in the stack (prep np pp ....). The action *subj* fills the slot subject with "the book" in the structure in top of structs, g01.
Line 2: *Coalesce* matches the non-terminal *np* to "peter" returning true. The action corresponding to that non-terminal, namely *rel-np*, is executed. This action creates the slot rel in g01 and fills it with "the book," for use later; pushes "subj verb +" in the non-terminals stack; and creates a new structure, say g02, and pushes it on *structs*. The stack *structs* contains (g02 g01). Line 3: Coalesce matches the non-terminal *subj* to "peter" returning true. The action for *subj*, after calling the noungroup function, builds the slot subject in the structure in top of structs, namely g02, and fills it with "peter." Line 4: Coalesce does not match any of the arcs to "donated" and returns false. The main algorithm checks to see if * is one of those arcs and, consequently, pops the stack. Line 5: Coalesce returns true. The action for *verb* opens the verb slot in top of *structs* and fills it with "donated," and pushes the usage for "donated" on the stack. Line 6: Coalesce returns false since it cannot match "obj" to "vanished." The main algorithm checks for the non-terminal obj on the top of the stack. Since that is the case, it then searches the structures being built for a slot called rel. In this case, it finds it in the structure g01. Then, it makes the content of slot rel the object of the structure being built (g02); changes the name of the slot from rel to rela, so that it cannot be used again; and fills the rela slot with the name of the structure in top of structs, namely g02. At this point the contents of the structures are:

```
g01 = (subject (the book) rela (g02))

g02 = (subject (peter) verb (donated) object (the book))
```

Line 7: Coalesce returns false. The main algorithm pops the stack since the asterisk belongs to the non-terminals in the top of stack. Line 8: When the main algorithm finds + on top of *structs*, it pops the non-terminals stack and the *structs* stack. Then, the gensym g02 is popped from structs, which is left only with g01. Line 9: Coalesce matches *verb* to

"vanished" and the action for *verb* creates and fills the slot verb with "vanished." Note that the structure is g01, since g02 has been popped. The structure g01 will contain at that point:

```
g01 = (subject (the book) rela (g02) verb (vanished))
```

Line 10: the structure g01 is popped from the stack. Line 11: the dollar sign and the end of the sentence are reached, consequently accepting the sentence.

3 The Acquisition of the Syntactic Usages

One easy way to define a syntactical usage for a new word is to search the lexicon for another word which behaves the same as the word one is defining and to copy its definition into the new one. This method, of course, is limited by the word usages currently defined in the lexicon. Moreover, there are many words which do not fall into any category. An even more serious problem with this method is that most end-users are unable to determine if a word has the same syntactical distribution as another one.

It is clear that if one wants to help end-users it is imperative that the usage be built from sample sentences posed by the users themselves. In the case of verbs, one could think of attacking this problem by attempting to parse the user's sentences with all distinct verb usages in the lexicon, and picking that usage which succeeds for all sample sentences. Again this method will be limited by the usages presently defined in the lexicon, but a more serious limitation with this method is that it simply does not work. This may be illustrated with a very simple example. Let us assume that a user tries to define the verb "read" and he/she types the sample sentences "Mary read" and "Mary read the book." Both sentences will be successfully parsed with the following usages (among others): (obj *), (obj) and (obj (inf *)). Of these three usages only the first one is correct. The interface could rule out some of these usages by generating sentences from the successful usages and asking the user if the sentences are grammatical. For instance, the interface could generate the following sentence from the third usage: "Mary reads Peter to go to the movies" and ask the user whether the sentence is correct. This method could work for some cases. However, it may produce a tedious interaction with the user. It also has the problem that a user may reject a sentence as ungrammatical for semantic reasons rather than for syntactic reasons. Most people will consider "rocks eat ideas" as ungrammatical although the sentence is syntactically correct. The solution to this problem is to parse the user's sample sentences with pieces or segments of usages. What a segment usage is will be discussed later. For now, let us observe that the usage (inf (obj (inf *))) is composed of the following segments "obj," "inf," "obj inf." This is to say a verb may be followed by object, or infinitive, or object plus infinitive. Of course, a verb may be followed by many other things which will be discussed in the next section. The basic idea is to parse the user's sentence with these segment usages, save all segment usages which successfully parse the sentence and finally merge those usages into the final usage for the word. The algorithm is described in Table 1. Suppose that a user is defining "read" and he/she types the sentences "Mary read" and "Mary read a book," and let us further assume that the segment usages in the list Segment-Usages are the ones

0. Repeat steps 1 to 4 until the user stops entering sample sentences.

1. Input user's sample sentence.

2. Select segment usages by examining the senses of the word immediately following the one the user is defining.

3. Put these segments in the list Segment-Usages.

4. Repeat steps 4.1 to 4.3 until the list Segment-Usages is empty.

4.1 Parse the user's sentence with the first usage in the list Segment-Usages.

4.2 If the parser succeeds, save the usage in the list Successful-Segment-Usages.

4.3 Remove the first usage from Segment-Usages.

5. Merge the segment usages in Successful-Segment-Usages into a single usage.

Table 1: *Algorithm for selecting the relevant usages of a word*

discussed above. In this case, the parser will succeed when it parses the sentences with (*) and (obj) as usages of "read". The list Successful-Segment-Usages will contain ((obj) (*)) in step 5 of the algorithm. This list will be passed to the function *merge* which will form a single usage for "read" *based on the two sentences* provided by the user. Let us consider each one of these steps separately.

4 Segment Usages for Verbs

First, we discuss the acquisition of the syntactical usage for verbs. The list below contains the segment usages for verbs, which are implemented in the current version of the interface, and also example sentences illustrating each usage. All these segment usages are stored in the list Segment-Usages. For each one of these segments, there is a corresponding action which is not listed here.

1.	(*)	He read.
2.	(adj)	He looks sad or He is sad.
3.	(obj)	He read the book
4.	(obj adj)	He kept her warm.
5.	(obj bare-inf)	I saw him fall.
6.	(obj inf)	I want Mary to read a book.
7.	(obj ing)	I saw her reading a book. (obj + present particple)
		I stopped him talking all night. (obj +gerund)

```
8.  (obj io)                 I gave Mary the book.
9.  (obj verb)               I know Mary likes Peter.
10. (obj adverb inf)         She showed him how to read.
11. (obj particle)           He gave smoking up.
12. (obj adverb subj verb)   He showed Mary how Peter proved the theorem.
13. (inf)                    Peter wants to read.
14. (prep)                   She believes in you.
15. (ing)                    Peter likes flying planes.
16. (poss ing)               I enjoyed his coming.
17. (adverb inf)             He knows how to read.
18. (adverb subj verb)       He knows why Mary likes books.
19. (that subj verb)         He knows that Mary likes books.
```

The segment usage (obj inf), as has been explained, will succeed in sample sentences like "I want Mary to read." Note that that usage will fail in the sentence "I want Mary" because "Mary" is not followed by an infinitive. The usage (obj bare-inf) is handled like the (obj inf). The NP following the verb is taken as a direct object, but if a bare infinitive is found, the obj is made the subject of the bare infinitive. The segment usage (obj io) handles bitransitive verbs by the usual procedure of assuming that the NP following the verb is the object, but if that NP is immediately followed by another one, the action for the non-terminal io will change the object slot to io, and will make the second NP the object. That segment usage will not handle indirect objects introduced by the prepositions "to" or "for." The segment (obj verb) handles verbs which take a clause as direct object e.g., "Mary knows Peter likes her." This is a done in a way similar to the usage (obj inf). The parser will take the NP following the verb to be the object, but if a verb is found then the the object will become the subject of the verb. Note that it is the usage (obj (verb *)) which makes possible this parsing. The usage (obj (verb *)) is the result of merging the segments (obj) and (obj verb). The usage (obj particle) deals with particles. Prepositions and adverbs which can be particles are marked in the dictionary with the marker *particle*. Users are told to define phrasal verbs by entering the verb followed by the particle or particles. However, this did not work in some cases. For instance, while defining the verb "give," some users typed the sample sentence "They gave it up," which only can be handled properly if particles are part of segment usages. The segment usages (prep) and (ing) are for verbs which are followed by a preposition and a gerund. In the case of (ing), there is a problem with ambiguity. If a user is defining the verb "like" and she/he types the sentence "Mary likes flying planes," the parser will return an ambiguous parse for this sentence, and also for the sentence "Mary sells flying planes." Ambiguous cases are discussed in section 6. The two segments which start with "adverb" handle sentences like "Mary knows how to read books" and "Mary understood how Peter entered the house." Finally, the last segment covers that-clauses like "I think that Peter saw Mary."

There is no need to try a user's sample sentence defining a verb with all segment usages, since the word immediately following the one she/he is defining constrains the relevant usages to a small subset. When a user types a sample sentence the senses of the word following the one being defined are examined. If the following word is unknown, he/she is

1. **Make the list Successful-Segment-Usages a set.**

2. **Move all singleton subsets of Successful-Segment-Usages to the list Singleton-Sets.**

3. **Repeat steps 3.1 up to 3.4 until the set Successful-Segment-Usages is empty.**

   3.1 Call the first subset of Successful-Segment-Usages FIRST. Move FIRST and all subsets of Successful-Segment-Usages whose first element is the same as the first element in FIRST to the list L.

   3.2 If there is more than one set in L, eliminate the first element of all subsets of L, except from one. Form the union of all subsets of L and put the result back in L.

   3.3 If the first element in the first subset of L is a member of any set in Singleton-Sets, remove that set from Singleton-Sets and insert-the-asterisk in L. (The function insert-the-asterisk yields (a (b c d *)) if L is equal to (a b c d). In Lisp this is defined as (list (car l) (append (cdr l) '(*))).)

   3.4 Put L in Final-Usage. Set L to nil.

4. **Move all sets in Singleton-Sets - if any left - to Final-Usage.**

5. **Return Final-Usage as the final usage for word.**

Table 2: *Algorithm for merging the relevant usages of a word into a determinitic representation*

asked to define that word prior to proceeding with the one presently being defined.

5 Merging the Segment Usages into a Final Usage

For the sample sentences "Peter wants a book," "Peter wants Mary to buy a book" and "Peter wants to buy a book," the parser will succeed with the following segment usages: (obj), (obj inf) and (inf), respectively. When the function *merge* is called it will merge those segment usages into the final usage for "want," namely (inf (obj (inf *))). The role of *merge* is to transform a representation which could be highly non-deterministic into a deterministic one, making it possible for *coalesce* (see appendix) to select deterministically the correct arc. The algorithm for *merge* is described in Table 2. (Successful-Segment-Usages is a list containing all segment usages which successfully parsed the user's sample sentences). Let us explain each one of these steps and illustrate them with examples. Let us assume that Successful-Segment-Usages contains ((a) (a f) (a k) (a m) (h m) (d)). Step 1 is necessary because two or more sample sentences can be parsed using the same usage.

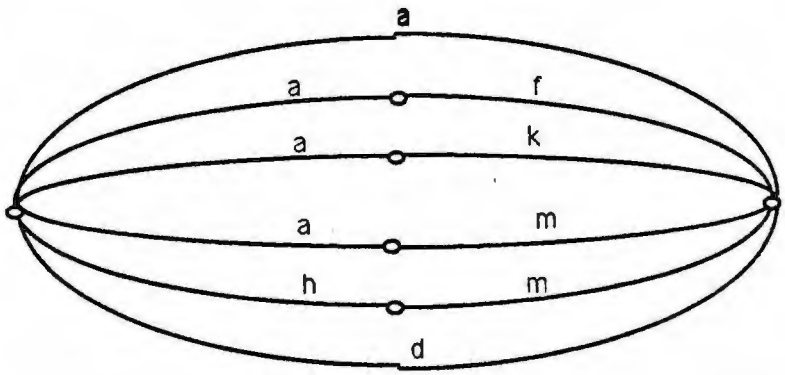


Figure 2: *Transition diagram representation of input ((a) (a f) (a k) (a m) (h m) (d)) for the function* merge

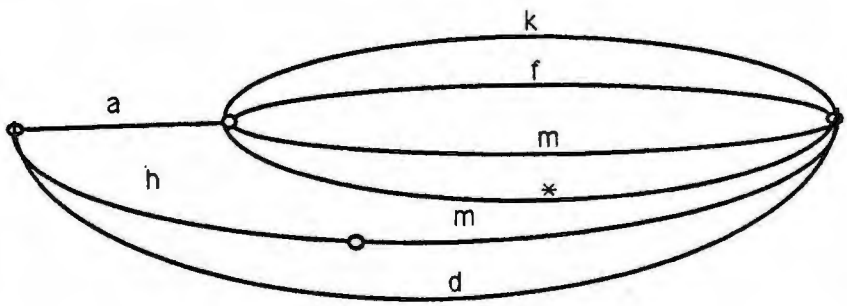


Figure 3: *Transition diagram representation of merge's output for input depicted in Figure 2.*

For instance, a user may type "Peter wants the red book" and "Peter wants books" as examples of two usages of "wants." In that case, the list Successful-Segment-Usages will contain (obj) twice. We could tell the user that his/her second sample sentence does not show a new usage of "wants," and teach her/him a little bit of syntax along the way. We may do that in later versions of the interface. Step 2 is intended only to facilitate the next two steps. After executing step 2, the list Successful-Segment-Usages is equal to ((a f) (a k) (a m) (h m)) and the list Singleton-Sets is equal to ((a) (d)). After executing 3.1 the list L is equal to ((a f) (a k) (a m)) and the list Successful-Segment-Usages is equal to ((h m)). After executing step 3.2, L is equal to (a f k m). After executing step 3.3, L is equal to (a (f k m *)) and Singleton-Sets is equal to ((d)). After executing step 3.4, Final-Usage is equal to ((a (f k m *))) and L is equal to nil.
In the second iteration (h m) is put into Final-Usage. The asterisk is not inserted because (h) does not belong to the Singleton-Sets. In step 4, (d) is moved to the Final-Usage whose final content becomes ((a (f k m *)) (h m) d). Figure 2 depicts the example above in a transition diagram, and Figure 3 depicts the output of *merge*.

6 Acquiring Usages for Nouns and Adjectives

It is very easy for an end-user to understand that in order to define a verb she/he needs to type some sample sentences using the verb. However, a completely different situation occurs in the case of nouns and adjectives. We mention that some nouns and adjectives have special "constructions" and provide some examples. This does not work in many cases, however, because many end-users do not distinguish between "that" as a conjunction and as a relative. It is common for a user to define, say, "announcement" or "ability" as a noun without providing any sample sentences indicating its usage. Then, one week later he/she is surprised that the parser fails to parse the sentences "The announcement that Peter was coming pleased her" and "Her ability to write is unique." The problem with reminding users that certain nouns may be followed by a *that-clause* is that the next time they are defining a noun, say, "news," they may type the following sample sentence "The news that Peter brought was sad." This does not cause any problem for the interface, because it is very easy to prevent this type of mistake by parsing the sentence with "that" as a relative in case the other usages for nouns fail.
This problem is even worse in the case of adjectives, because some adjectives in combination with *be* distribute over a large domain of different usages. The limited experience we have had with the interface has indicated to us that the best way to deal with this problem is by telling the users not to be overly concerned about the definition of adjectives or nouns until the parser fails to parse a sentence. Then, they should activate the interface with the *same* sentence that failed (The automatic activation of the interface after the parser fails has not yet been implemented.). Because the user's sentence may contain several user-defined words, they must redefine the one that they suspect caused the problem An aspect that is presently under design is to let the parser determine which word caused the problem and correct it by automatically activating the interface with the failed sentence. The following three usages for nouns are implemented:

```
(inf)                  Peter has the ability to read fast.
(that-clause)          The fact that Mary came pleased him.
(it be np subj verb)   It's a pity Kafka died so young.
```

In the third usage, the head noun of the np (*pity*) is the *noun* being defined by the user. The following usages for adjectives have been implemented:

```
(be adj inf)           It was impossible to read at the store.
(be adj prep np inf)   It was nice of you to help him.
                       It was impossible for Peter to leave.
(be adj that-clause)   I am sad that Mary left.
(be adj prep)          I am sorry about your tree.
```

7 Ambiguity and Other Problems

In his/her sample sentence a user may include any word which is defined in the lexicon, or he/she may define a word and then use that word in defining other words. For instance, a user may define "duck" by saying that it is a noun and a verb. Then, in defining the verb "see," he/she may type the sample sentence "I saw her duck." The interface will try this sentence with all segment usages which begin with obj. The parser will succeed with the three segment usages (obj), (obj bare-inf) and (obj io). A user may type the sample sentence "I took her flowers" for which the parser will succeed with the segments (obj) (obj io). In an earlier version of the interface, if a user's sentence was ambiguous he/she was asked to choose another sample sentence. In our present implementation, the user's sample sentence is modified and he/she is asked to judge if the altered sentence is grammatical. The modification of the sentences follow very simple rules. A lexical entry which may be a pronoun or an adjective is replaced with a proper noun. For instance, "her" will be replaced in "I saw her duck" with "Mary" resulting in "I saw Mary duck." Then, the user will be asked if this sentence is grammatical. In the sentence "I like flying planes," the parser will succeed with the segment usages (obj) and (ing). In this case, the rule used is to modify the user's sentence by deleting every word following the -ing *form*. In this example, the altered sentence will become "I like flying."

The unrestricted use of adverbs by end-users caused some problems in an earlier stage of the interface. The problem was solved by marking in the dictionary those adverbs and prepositions which may act as particles, and also by encoding knowledge in the parser about a large class of adverbs including words such as "yesterday," "later," "there," "home," ... Time NPs may also pose some problems because they may appear almost anywhere in the sentence. For instance, a user may type a sentence like "Popeye ate spinach every day," which will result in building an indirect object for "ate." Of course, this problem can easily be solved by asking the users not to construct sample sentences having nouns referring to time, such as "day," "hour," etc. Because the present implementation of the parser is able to handle time NPs, we are asking the user to tell us if the noun they are defining is a time noun. If they do that, the parser is able to determine that "every day" in the sentence above is a time NP.

8 Summary and Conclusions

We have presented an algorithm which acquires syntactic rules for verbs, nouns and adjectives from sample sentences entered by end-users. The algorithm synthesizes the rules in a way which allows the parser to parse the sentences deterministically. The interface is now being used extensively by SNOWY's users, and has performed beyond our most optimistic expectations. It is helping us to gather highly interesting information about naive-users' intuitions on the usage of English grammar.

The theory which underlies our approach to syntactic knowledge can be traced back to a paper by Gross [Gross, 79]. Gross and his associates realized, after building a transformational-generative grammar of French containing 600 rules; that the task could not

be achieved without abandoning the main theoretical tenets of transformational-grammar. This realization came about after applying these 600 rules to over 12,000 lexical items. Gross found out that for every syntactical transformational rule one can find some lexical items which obey the rule and others which don't. In fact, he concluded that "for French, ... the phenomena that would be called lexical by Chomsky are the rule, while the ones he termed transformational are quite rare." (See [Gomez, 1989] for a more extensive discussion of these issues). Our syntactic knowledge acquisition algorithms are a consequence of this point of view: "General" syntactic rules are not the target of the acquisition algorithms, but rather verb patterns, adjective patterns, etc. This is clearly in contrast with the ideas presented in [Berwick, 84].

From a learning point of view, our approach is a knowledge-intensive one. Our learning algorithms learn because they already know a lot. This knowledge is in the parser which covers a considerable portion of English syntax, in the form of a few "general" syntactic rules and a considerable number of syntactic word usages.

Finally, the way in which our algorithm acquires the syntax of new words is somewhat similar to the way in which a non-native speaker of a language acquires the syntax of new words. He/she, we hypothesize, starts by mastering a set of basic and "general" syntactic rules (e.g. passive, relative clauses ...) and, then, he/she begins to face the enormous irregularity of English syntax by learning the syntax of individual words. Then, one day he/she starts building sentences with the verbs "want" and "expect" such as:

Mary expected to read a book.
Mary wanted to read a book.

Then, when in future days she/he says,

Mary was expected to read a book.
***Mary was wanted to read a book.**

he/she finds with the unpleasant surprise that "Mary was wanted to read a book" is not grammatical, and that she/he needs to learn a usage for "want" and another for "expect." Now, when she/he had thought that the two verbs could be put in "the same sack"!

References

Ballard, B. and Tinkham, N.(1984) A grammatical framework for transportable natural language processing. *Computational Linguistics*, 10, 2, 81-96.

Berwick, R. C. (1985) *The Acquisition of Syntactic Knowledge.* MIT Press, Cambridge, Massachusetts.

Carter, D. (1989) Lexical acquisition in the core language engine. CRC-012, Cambridge Research Center, SRI International.

Gomez, F. (1988). WUP: A parser based on word usage, in *Proceedings of the 1988 IEEE Conference on Computers and Communication,* Scottsdale, Arizona, 445-449.

Gomez, F. (1989). WUP: A parser based on word usage. UCF-CS-89-1, Department of Computer Science, Orlando, Fl 32751.

Gomez, F. and Segami, C. (1990) Knowledge Acquisition from Natural Language for Expert Systems Based on Classification Problem-Solving Methods. *Knowledge Acquisition*, 2, pp. 107-128.

Gomez, F. and Segami, C. (1989) The Recognition and Integration of Concepts in Understanding Scientific Texts. *Journal of Experimental and Theoretical Artificial Intelligence.* Vol.1, pp. 51-77.

Gomez, F. and Segami, C. (1991) Classification-Based Reasoning. *IEEE Transactions on Systems, Man and Cybernetics* (June-July issue, to appear).

Gross, M. (1979) On the Failure of Generative Grammar. *Language* Vol. 55, No. 4, 859-883.

Grosz, B., Appelt, D., Martin, P. and Pereira, F. (1987) TEAM: An experiment in the design of transportable natural-language interfaces. *Artificial Intelligence,* v. 32, (2), 173-243.

Appendix

Sample Session with the Interface

```
Script started on Sun Oct 14 00:57:41 1990
[1] lisp

Franz Lisp, Opus 38.92

->
(include interface.l]
[load interface.l]
t
->
(parser) REMARKS: PARSER IS ACTIVATED
-> type your sentence.
they believed in mary per
-> (subj verb + $)->(they believed in mary)
-> (i do not understand * believed *)
-> ((prep np pp rel ing-rel *) verb + $)->(believed in mary)
-> (verb + $)->(believed in mary)
-> fail to parse sentence.   REMARKS: PARSER FAILS BECAUSE IT
                              DOES NOT KNOW "BELIEVED"
-> parsing-attempts-completed
```

```
(interface) REMARKS: INTERFACE IS ACTIVATED
-> Please tell me if the word you want to add is a verb. yes/no?
yes
-> Please enter in a list the present tense, the simple past and the past
-> participle forms of the verb. E.g. (read read read)
(believe believed believed)
-> Please enter a sentence using the SIMPLE PAST.
they believed in mary.
-> Do you want to type another sentence using the same word, but showing a
different use of the word? yes/no?
-> ye
-> Please write yes/no.
yes
-> Please enter a sentence using the SIMPLE PAST.
they believed mary.
-> Do you want to type another sentence using the same word.
yes
-> Enter sentence using the SIMPLE PAST.
they believed mary liked peter.
-> Do you want to type another sentence? yes/no?
yes
-> Enter sentence using the SIMPLE PAST.
they believed that mary liked peter.
-> Do you want to type another sentence? yes/no?
yes
-> Enter sentence using the SIMPLE PAST.
they believed that man.
-> Do you want to type another sentence? yes/no?
no
-> Besides ** in ** which other preposition may follow "believe"?
-> if only those prepositions write "only"
-> if most prepositions may follow it, write "most,"
-> if other prepositions may follow it, write those prepositions
-> in a list
-> help/most/only/list containing other prepositions?
only
-> Do you want to store the new word in the file user-lex? yes/no?
yes
-> Do you want to add another word?yes/no?
no
(parser) REMARKS: PARSER IS ACTIVATED AGAIN
-> type your sentence.
they believed mary.
```

```
-> (subj verb + $)->(they believed mary)
-> ((prep np pp rel ing-rel *) verb + $)->(believed mary)
-> (verb + $)->(believed mary)
   REMARKS: NOTES THE USAGES FOR ''BELIEVE''
-> ((in (obj (verb *)) that) + $)->(mary)
-> ((prep np pp rel ing-rel *) (verb *) + $)->nil
-> ((verb *) + $)->nil
-> (+ $)->nil
-> ($)->nil
-> parsed! want to see structures? y/n?
n
-> parsing-attempts-completed
-> type your sentence.
they believed mary went to the barn.
-> (subj verb + $)->(they believed mary went to the barn)
-> ((prep np pp rel ing-rel *) verb + $)->(believed mary went to the barn)
-> (verb + $)->(believed mary went to the barn)
-> ((in (obj (verb *)) that) + $)->(mary went to the barn)
-> ((prep np pp rel ing-rel *) (verb *) + $)->(went to the barn)
-> ((verb *) + $)->(went to the barn)
-> ((to inf) + + $)->(to the barn)
-> ((prep np pp rel ing-rel *) + + $)->nil
-> (+ + $)->nil
-> (+ $)->nil
-> ($)->nil
-> parsed! want to see structures? y/n?
y
-> used-structs
g00022
  REMARKS: STRUCTURES BUILT BY THE PARSER
-> (setq tep '(subj ((pron they)) verb (believed) obj g00025))
-> g00025
-> (setq tep
      '(subj ((pn mary)) verb (went) prep (to ((dfart the) (noun barn)))))
-> type any letter to continue.
done

-> parsing-attempts-completed
-> type your sentence.
done
->
Goodbye
[2] exit
```

An algebraic method for inducing efficient decision trees

Daniel Jaliff
Department of Computer Science, University of Calgary
2500 University Drive N.W., Calgary, Alberta
Canada T2N 1N4
jaliff@cpsc.ucalgary.ca

Abstract An efficient algorithm for reducing decision trees by structural manipulation is shown to be a suitable platform for an inductive learner which compares favorably with a system that induces decision trees using statistical heuristics.

1. INTRODUCTION

Existing systems geared toward the induction of concepts from sets of training examples, such as ID3 [Quinlan, 1986a] and PRISM [Cendrowska, 1987], use statistical heuristics to arrive at their concept definitions.

This paper is an attempt to characterize the problem of data-driven induction -- concept induction from a fixed set of a training examples -- as one that can be solved using algebraic operators, without exploiting statistical measures of the training set.

Such a solution does not require the assumption, made by statistically-based systems, that the training set distribution is representative of the population distribution. It is also more transparent than ID3 in the way it maintains of logical constraints during the induction process. Unlike ID3, it is consistent with the view of induction as a *process* of generalization from a set of examples.

A brief overview of ID3 is followed by a description of a tree reduction algorithm that uses structural manipulation of the nodes [Cockett and Herrera, 1990].

The system Al (Algebraic Learner) is presented in terms of the modifications and additions to the tree reduction algorithm required for this application. Preliminary results are presented and compared to results obtained using ID3 on the same training set.

2. A STATISTICAL METHOD: ID3

ID3 receives as input a training set S composed of events described by a set of attributes A. Each event is represented as a vector $<a_1,a_2,...,a_n,c_k>$ where$\{a_i\}$ are attribute values from discrete domains and c_k is a unique class associated with $<a_1,a_2,...,a_n>$.

For each a_i, ID3 calculates the average entropy [Cendrowska, 1987] of the subsets resulting from a partition of S by the each vector's value on a_i. The systems then selects the attribute a´ that minimizes this average entropy and creates a decision tree with a´ at its root.

This procedure is applied recursively to the subsets resulting from the partition of S by a´. The algorithm terminates when for every a_i, the entropy of the subsets that would result from partitioning S by a´ is zero -- indicating that all members of S belong to the same c_k.

Since every event in S must be tested on all of its attributes at every level of the tree, the complexity of this algorithm is $O(|A|^2 |S|)$.

a	b	c	d	δ	a	b	c	d	δ	a	b	c	d	δ
1	1	1	1	3	2	1	1	1	3	3	1	1	1	3
1	1	1	2	2	2	1	1	2	2	3	1	1	2	3
1	1	2	1	3	2	1	2	1	3	3	1	2	1	3
1	1	2	2	1	2	1	2	2	1	3	1	2	2	1
1	2	1	1	3	2	2	1	1	3	3	2	1	1	3
1	2	1	2	2	2	2	1	2	2	3	2	1	2	2
1	2	2	1	3	2	2	2	1	3	3	2	2	1	3
1	2	2	2	1	2	2	2	2	3	3	2	2	2	3

a:	**1. young**	**2. pre-presbyopic**	**3. presbyopic**
b:	**1. myope**	**2. hypermetrope**	
c:	**1. not astigmatic**	**2. astigmatic**	
d:	**1. reduced tear production**	**2. normal tear production**	
δ:	**1. fit hard contact lenses**	**2. soft contact lenses**	**3. none**

Table 1 - Contact lens data.

The information-theoretic heuristic that is essential in guiding ID3 in the formation of non-trivial concepts obscures the maintenance of logical constraints that characterizes the inductive process. To see why this is the case, let us first review a formal definition of induction [Genesereth and Nilsson, 1987].

Let Γ be a set of statements in logic which constitute a body of domain knowledge. Let Δ be a database of newly acquired information about the domain, and let ψ be a hypothesis. We say that ψ is an inductive conclusion of $\Gamma \cup \Delta$ if the following conditions hold.

1. ψ is consistent with Γ and Δ.

2. Δ is provable from Γ and ψ.

A logical characterization of ID3's recursive step is not possible because many of the resulting tree's leaves may be labeled with the same class; the partition of the training set Δ does not guarantee that members of a specific class will all be classified by a single subtree of the last selected attribute. The only "hypothesis" made is that the selected a' will be referenced in all of the class definitions to be induced on the basis of the portion of Δ assigned to the current node.

Table 1 contains Cendrowska's complete training set for the prescription of contact lenses, and Figure 1 shows ID3's output on the data [Cendrowska, 1987].

While ID3's output is indeed a generalization of the training data, the process whereby this result is obtained is one of specialization: the algorithm's recursive step is the search for a more specialized definition that will bring more "order" to the assigned portion of the training set.

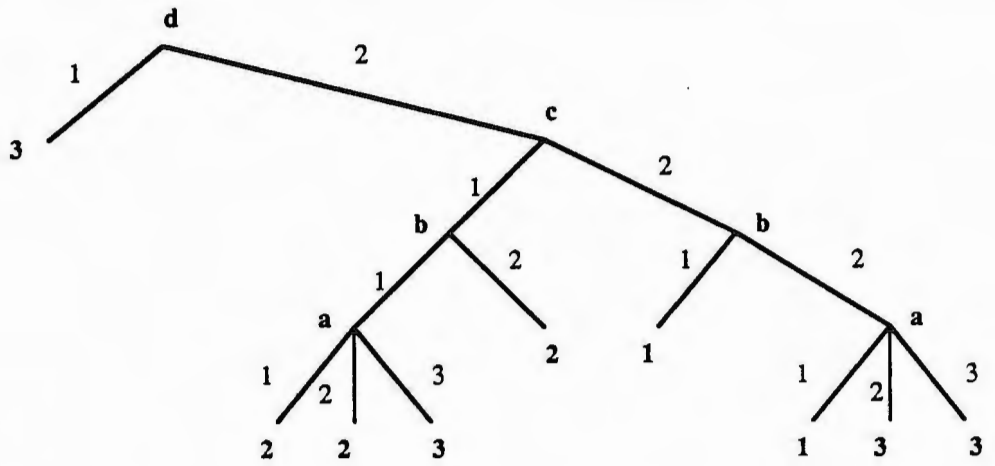


Figure 1 - ID3's output on the contact lens data.

3. TREE REDUCTION

This section presents the tree reduction algorithm of Cockett and Herrera, including some theoretical preliminaries [Cockett and Herrera, 1990].

3.1 Decision theories

A *decision theory* D is a pair (A,E), where $a \in A$ is a *test* , or *attribute* , with *arity,* or *branch factor* greater than one, and $e \in E$ is an equation of the form T=T'. T and T' are *terms*, or *trees*, defined as follows:

- If $T \in X$, where X is a set of *variables*, or *classes*, T is a tree.

- If $T_1,...,T_{arity(a)}$ are trees, then $T=(a\ (T_1,...,T_{arity(a)}))$ is a tree, for all $a \in A$.

A decision theory can be viewed as the set of all decision trees that can be built from attributes A and decisions X.

In addition to the equations E, the trees of D must satisfy the following identities:

1. Idempotence:

$$(a\ (T,...,T)) = T.$$

2. Repetition:

$$(a\ (T_1,...,T_{k-1},(a\ (T_{k1},...,T_{kn})),T_{k+1},...,T_n) = (a\ (T_1,...,T_{k-1},T_{kk},...,T_n)).$$

A tree whose every path from the root to a leaf is free of repeated tests is said to be *repeat reduced.* A repeat reduced tree to which idempotence reduction has been applied wherever possible is said to be *simply reduced.*

3. Transposition:

$$(a_1\ ((a_2\ (T_{11},...,T_{1m})),...,(a_2\ (T_{n1},...,T_{nm})))) =$$
$$(a_2\ ((a_1\ (T_{11},...,T_{n1})),...,(a_1\ (T_{1m},...,T_{nm})))) .$$

These identities are shown graphically as operators in Figure 2. It is clear that these operators preserve the tree's semantics. The idempotence, or *idempotence reduction* operator, can be viewed as a generalization step.

Two trees are said to be *decision equivalent* if one can be obtained from the other by a composition of these three operators. The trees are said to be *transposes* if one can be obtained from the other by repeated applications of the transposition operator only.

3.2 Irreducible trees

Let every path from the root of a tree to a leaf be called a *spine*, represented as an ordered list of test-outcome pairs. A tree can then be represented by its *spine set.*[1]. Two spine sets are equivalent if they belong to the same tree; they are *bag equivalent* if, when the spines are taken to be bags instead of ordered lists, they are the same spine set.

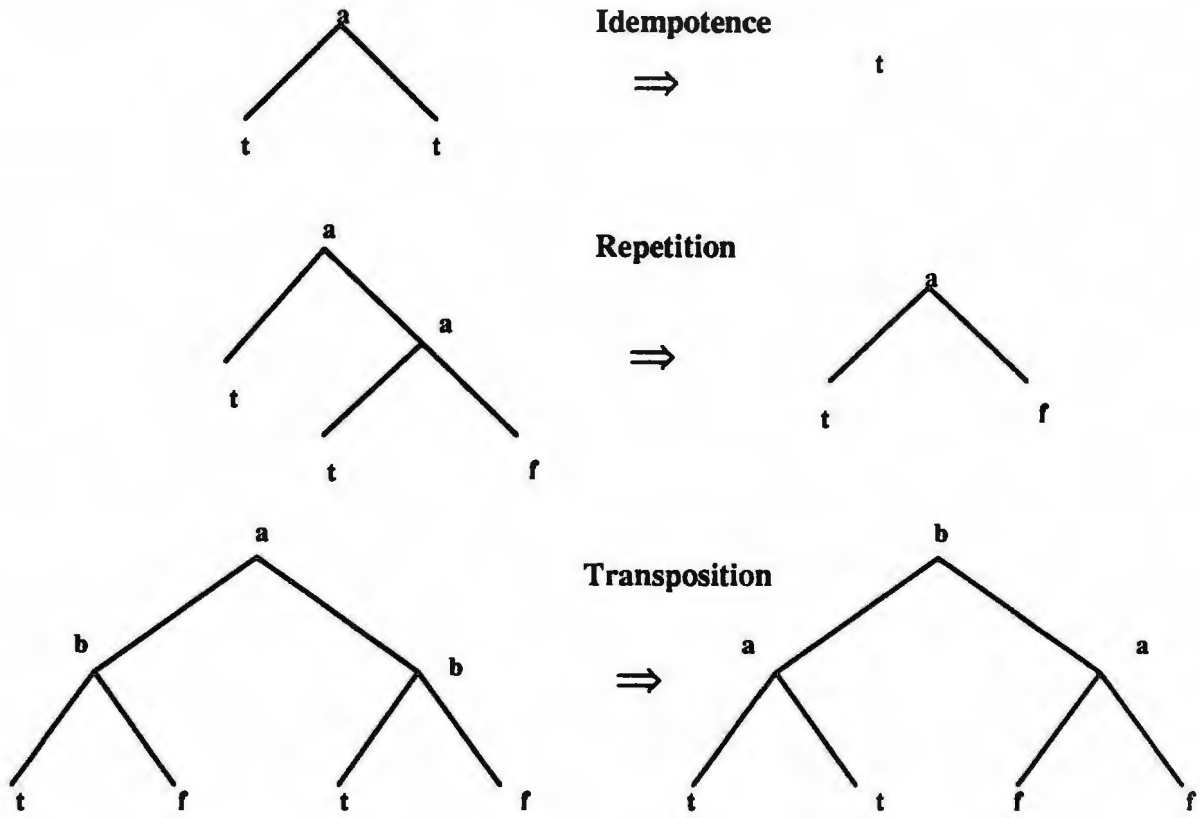


Figure 2 - Operations on the trees of a decision theory.

The Chen and Ras lemma [Chen and Ras, 1985] states that two repeat reduced trees have bag equivalent spine sets **iff** they are transposes. It is not difficult to show that if two trees are transposes, then they have bag equivalent spine sets. The converse proof is based on the observation that the transposition operator can be used to reshape one of the tree's spine sets to be bag equivalent to the other tree's spine set.

---

[1]This is equivalent to trivially extracting rules from a tree.

A tree is said to be *irreducible* **iff** all of its transposes are simply reduced; the tree reduction algorithm in the next subsection is based on this definition. What remains to be examined is the set of properties that can be expected of irreducible trees.

A *preorder* $\leq_p$ is a relation that is transitive and reflexive but not necessarily antisymmetric. A *least preorder* containing a relation is obtained by computing the relation's transitive-reflexive closure.

The *reasonable preorder* $\leq_r$ on the trees of a decision theory is the least preorder $\leq_p$ that possesses the following properties:

1. $T_1 \leq_p T_2$ implies that (a (...,T_1,...)) $\leq_p$ (a (...,T_2,...)).

2. $T \leq_p$ (a (T,...,T)) : the idempotence reduction of a tree is smaller than the original tree in the preorder.

3. (a (T_1,...,T_{k-1},T_{kk},...,T_n)))) $\leq_p$ (a (T_1,...,T_{k-1},(a (T_{k1},...,T_{kn})),T_{k+1},...,T_n))) : the repeat reduction of a tree is smaller than the original tree in the preorder.

4. (a_1 ((a_2 (T_{11},...,T_{1m})),...,(a_2 (T_{n1},...,T_{nm})))) $\leq_p$ (a_2 ((a_1 (T_{11},...,T_{n1})),...,(a_1 (T_{1m},...,T_{nm})))): any transpose of a tree is smaller than the original tree in the preorder.

Note that since $\leq_p$ is not necessarily antisymmetric, two different trees may satisfy both $T_1 \leq_p T_2$ and $T_2 \leq_p T_1$.

A *reasonable criterion* is a real cost function on the trees of a decision theory that preserves the reasonable preorder. From the definition of a reasonable preorder it follows that a tree is irreducible **iff** it is a minimal element in $\leq_r$. Therefore, every irreducible tree is optimal with respect to some reasonable criterion.

While the reduction algorithm for finding irreducible trees to be discussed in the following subsection does not allow the predetermination of the reasonable criterion to be used for reduction (nor does it return the criterion used), it guarantees that the tree will be optimal with respect to *some* reasonable criterion.

Some examples of reasonable criteria are the height and number of leaves of a tree, a set of costs associated with each test or attribute, or any linear combination of the above.

3.3 The reduction algorithm

As defined in the previous subsection, a tree is irreducible **iff** all of its transposes are simply reduced. In order to show that a tree is irreducible, one must show that there is no attribute that can be manipulated to the leaves by transposition, and there be eliminated by idempotence reduction.

An attribute a is said to be semi-essential in a decision tree T if it appears on every path from the root to a leaf. T is called *semi-essentially simple* when T's root is its only semi-essential.

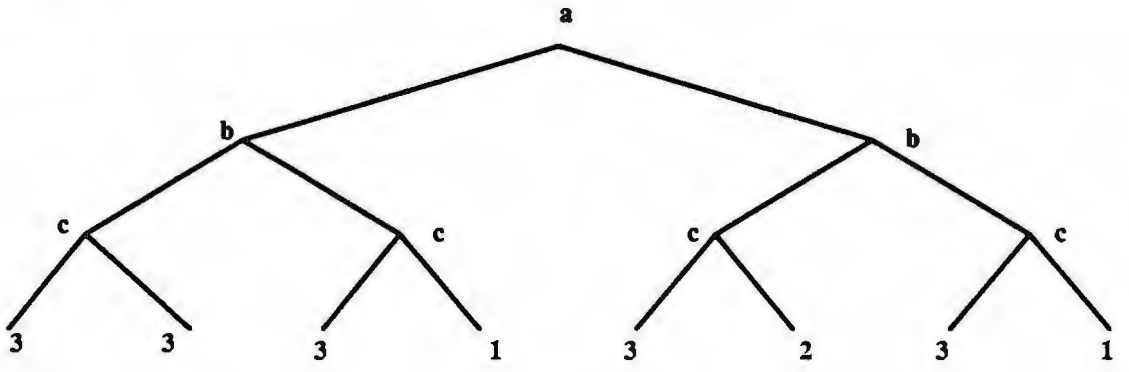
The tree is said to be factored by attribute a if every subtree with a at its root is semi-essentially simple. T can be factored by a by *pulling-up* all semi-essential attributes that appear below a, thus delaying test a.

```
reduce(T)
if T is a leaf, return T;
else apply "reduce" to every subtree of T,
       return (elimination-factor T).

elimination-factor(T)
if T is a leaf, return T;
else if all of T's subtrees are equal leaves,
       return the outcome under T;
else if T does not have any semi-essential tests
       (other than root(T)), return T;
else let a be some semi-essential in T (other than (root (T))),
       let T = pull-up(a,T) (pull a up to the root of T),
       apply "elimination-factor" to every subtree of T,
       return T.

semi-essentials(T)
if T is a leaf, return the empty set.
else return
       {root(T)} ∪ (intersection (map "semi-essentials" (subtrees T))).

pull-up(a,T)
return T' = (a (branch-reduce(a,1,T) (branch-reduce(a,2,T) ...
       ...(branch-reduce(a,arity(a),T))).

branch-reduce(a,i,T)
if T is a leaf return T;
else if a=root(T) return the ith subtree of T;
else apply "branch-reduce" with a and i to every subtree of T,
       return T.
```

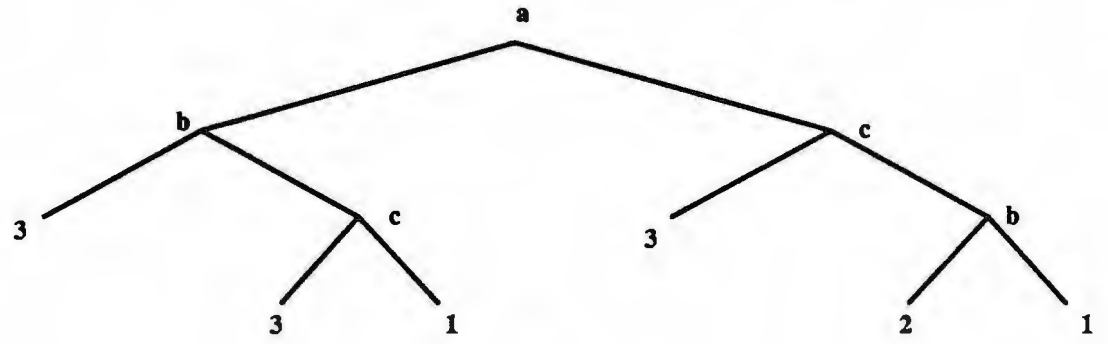
Figure 3 - The tree reduction algorithm.

A tree T is irreducible iff for every test a in the tree, T factored by a cannot be idempotence reduced [Cockett, 1987]. The reduction algorithm, shown in Figure 3, is based on this result.

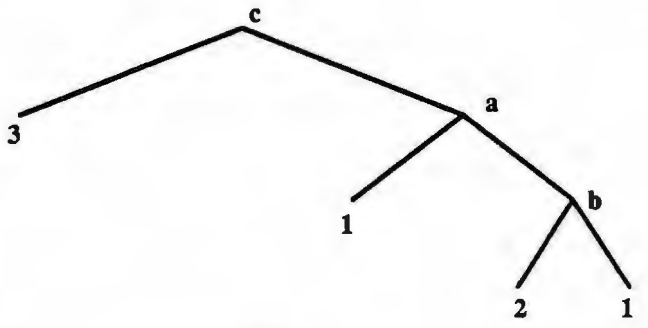
Let us show that that the algorithm is correct, given that the function `elimination-factor` factors a tree by the attribute at its root and performs idempotence reduction at the leaves.



(a)



(b)



(c)

Figure 4

If T is a leaf, or outcome, `reduce`(T) returns the outcome. Otherwise, assume that all of T's subtrees have been properly reduced. If T factored by the test at its root cannot be impotence reduced, then T with its subtrees reduced is irreducible, and the factoring will, at worst, transpose the tree.

If T factored by the root (T) *can* be idempotence reduced at the leaves, then it must be shown that the resulting tree is irreducible. Let us assume that it is not; then there exists an attribute a in T (a ≠ root(T)) such that T′ = `reduce`(T) can be factored by a, and the resulting tree T′′ can be idempotence reduced at the leaves. But T′′ is transpose equivalent to T factored *first* by a and then by root(T); the fact that T′′ can be idempotence reduced at the leaves contradicts the assumption that T's subtrees were converted to irreducibles.

Cockett and Herrera [Cockett and Herrera, 1990] show the complexity of this algorithm to be *O(height² leaves)*.

4. AL

The application of this tree reduction algorithm to inductive learning is straightforward; all that needs to be done is to build a "trivial" decision tree from the data and apply the reduction algorithm. This tree can be seen as a hypothesis that is consistent with the training set, and can be used to prove it. Since the algebraic operators used in reduction preserve the semantics of the tree, every intermediate operation maintains these logical constraints.

Even though the reduction algorithm does not accept as input the reasonable criterion with respect to which the output must be optimal, it is possible to make a slight modification that biases the algorithm toward the reduction of the number of leaves.

When passed the tree of Figure 4(a), the reduction algorithm returns the tree of Figure 4(b). The reasonable criterion by which this tree is optimal is far from obvious; we would like the reduced tree to be (or at least resemble) that of Figure 4(c).

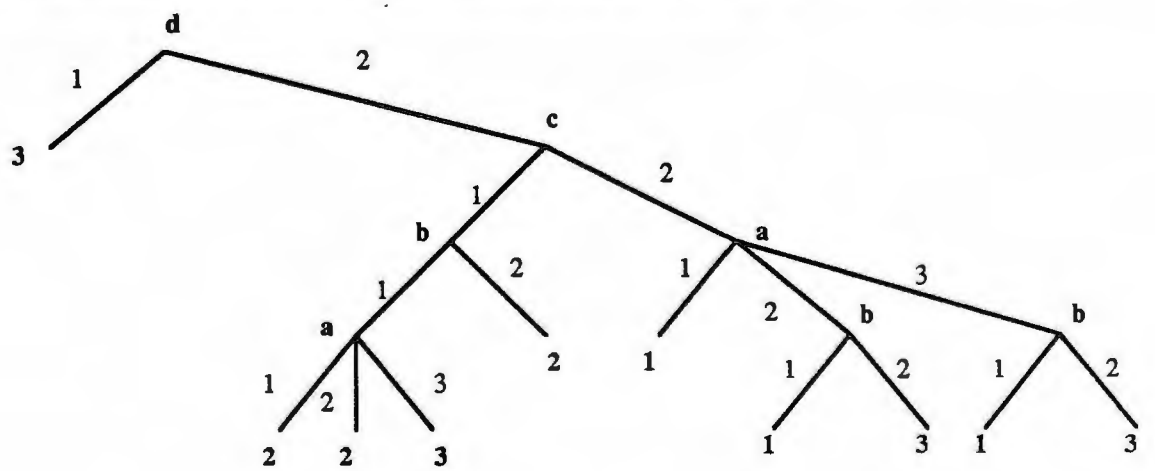


Figure 5 - Al's output on the contact lens data.

Suppose we wished to factor the tree of 4(b) by a. By the definition of semi-essential tests used so far, the left subtree has a single semi-essential, b, and the right subtree has the single semi-essential c. If the definition of `semi-essentials(T)` is loosened to include: "if all of T's subtrees but one are equal leaves, and that outcome is a child of the

non-leaf subtree, return the test at the root of the non-root subtree." This simple, inexpensive modification enables to reduce 4(a) to 4(c).

The complexity of Al is $O(|S|\,|A|)$ for building a trivial tree from the training set, and $O(|A|^2\,\Pi_{a \in A} arity(a))$ for the reduction. It follows for a given problem represented by an attribute set A, there exists an N such that for |S| > N, Al will perform better than ID3. In practice, the absence of floating point arithmetic operations in Al can be expected to make this N relatively small.

AL was tested on the contact lens data of Table 1; the resulting tree, almost identical to that produced by ID3, is shown in Figure 5. The Scheme implementation of Al, running on a SPARCstation 2, read the data file, built the tree, and reduced it to this form in 90ms.

4.1 Al and incomplete training sets

When building the full decision tree from the training set, Al marks unlabeled leaves with a "don't care" symbol. Incomplete training sets (or problems in which not all theoretically possible combinations of attribute values are present) will leave some (or many) of those "don't care" labels in place. These labels take the role of a wildcard in idempotence reduction. The implication is that for incomplete training sets (and *only* for such sets) Al does make some assumptions about relationships between the sample and population distributions.

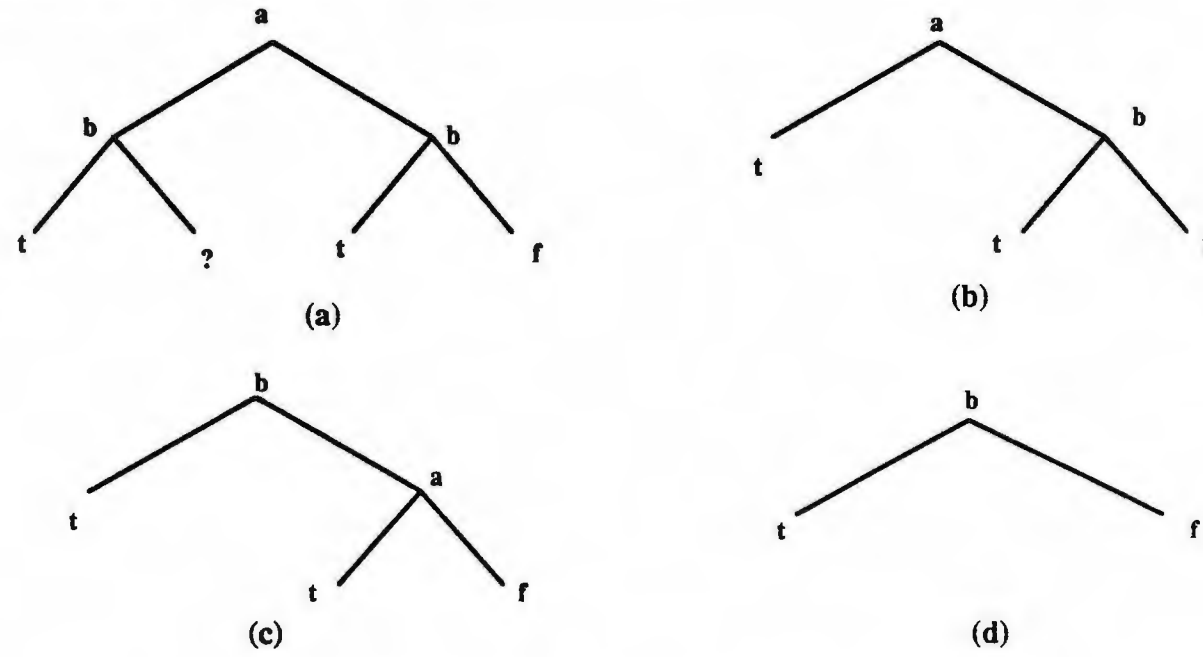


Figure 6 - Problems with the use of wildcards

The use of "don't care" labels gives rise to some implementation problems, as depicted in Figure 6[2]. The left subtree of 6(a) is reduced to "t", yielding 6(b). Then 6(b) is transposed to 6(c) by pulling up semi-essential test b. Simple inspection of 6(a) suggests that the desirable result is 6(d).

The solution is that when a node all of whose children are leaves is idempotence reduced, and at least one of the children was a wildcard, the reduced node should maintain information concerning its former children, for use when its previous test needs to be pulled up (Figure 7).

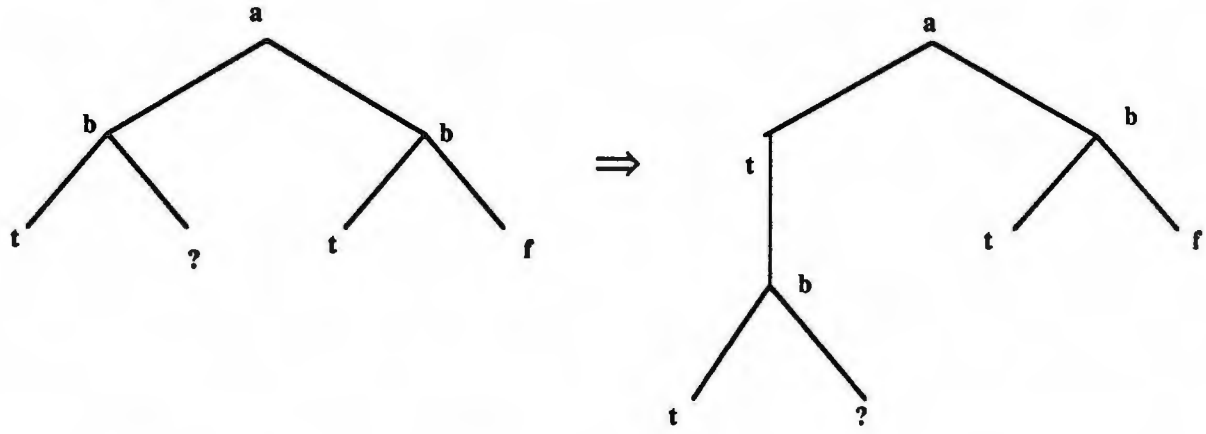


Figure 7 - Storing information about wildcard reductions.

The tree produced by Al from an incomplete training set is likely to have some leaves that remain labeled with a wildcard; and unseen event reaching that leaf would be classified as "unknown". This makes Al a conservative system, less likely than ID3 to make guesses on unseen events.

4.2 Al as an incremental learner

The system can be adapted easily to operate as an incremental learner. When an unseen instance that is not correctly classified by the existing tree, it is used to extend a path. It would only be necessary to perform reductions along this new path from the root to a leaf (the new instance) prior to performing factoring and idempotence reduction[3] of the entire tree. Alternately, the tree could be reduced only when the tree changes along a certain dimension (eg, the number of leaves) by a user-determined threshold.

4.3 Coping with noise

Al can be adapted to deal with noisy data, although of course some sort of statistical criterion would need to be applied

---

[2]The question mark symbol indicates a "don't care" leaf.

[3]That is, prior to applying `elimination-factor` to the entire tree.

Incorrect data would manifest itself in an ambiguous labeling of the leaves of the full tree constructed for input tot he reduction algorithm. In most cases it ought to be sufficient to label each leaf with the most frequent class that falls under that leaf, using the *majority method* described and tested by Quinlan [Quinlan, 1986b]. If this principle cannot be applied effectively for lack of a clear majority, this is probably a symptom of excessive noise, which might be solved by using a larger, or different, data set, or testing the events on additional attributes.

5. CONCLUSION

Al is an inductive system that learns concept definitions in the form of reduced decision trees by structural manipulation of the full tree built to match the training set. If the training set is complete, Al does not rely on heuristics; otherwise, it makes some statistically based assumptions.

The system is efficient and produces results comparable to those of ID3. It can be easily extended to cope with the presence of misclassified events in the training set, and constitutes a suitable platform for an inductive learner. Al is fully transparent in the way it performs generalization and maintains logical constraints during induction.

Acknowledgements

Many thanks to Robin Cockett, Brian Gaines, and Mildred Shaw for their helpful comments and suggestions.

References

Cendrowska, J. (1987) PRISM: An algorithm for inducing modular rules, *International Journal of Man-Machine Studies*, 27, 349--370.

Chen, K. and Ras, Z. (1985) Homogeneous information trees, *Funda. Inf. VIII*, 1, 123--149.

Cockett, J.R.B. (1987) Decision expression optimization, *Funda. Inf. X*, 93--114.

Cockett, J.R.B. and Herrera, J.A. (1990) Decision Tree Reduction, *Journal of the Association for Computing Machinery*, Vol. 37, No. 4, 815--842.

Genesereth, M.R. and Nilsson, N.J., *The logical foundations of artificial intelligence*, Kaufmann, Los Altos: 1987, 161--165.

Quinlan, J.R. (1986a) Induction of Decision Trees, *Machine Learning*, 1, 81--106.

Quinlan, J.R. (1986b) The effect of noise on concept learning, in Michalski, R. *et al* (eds.), *Machine Learning: an AI approach*, Vol II, Kaufmann, Los Altos: 1986, 149--165.

Experience in Applying KRS to an Actual Business Problem

Rob C. Kremer
Computing Research
Shell Canada Ltd.
P.O.Box 100, Station M
Calgary, Alberta, Canada
shellcr@cpsc.ucalgary.ca

Abstract

This paper traces the application of a knowledge representation server (KRS) to an actual business problem in financial transaction verification and correction. The problem was first attacked using a traditional expert system shell, but the translation of the business rules turned out to be verbose and difficult to understand. The Z specification language was applied in order to try to gain a better understanding of the problem domain, and this showed the need for rules with exceptions. KRS has rules with exceptions, a powerful "programming with pictures" visual language, a link to hypertext facilities and other innovative tools that apply to the problem. KRS has been applied with some success as a prototypical system. It is hoped that the simplicity of a visual language such as KRS's may enable the domain experts to directly input the original business rules and to handle the ongoing updates.

1. The Problem

The company's financial transaction records are created at the various business sites, then are batched into the financial and accounting programs. However, some of the transactions are rejected because they contain inconsistent or erroneous data. The system provides an error code with the rejection. Somehow, these rejected transactions must be corrected and then resubmitted to the accounting programs.

Currently, the corrections are being done by a group of people who rely on a book of rules indexed on the error code accompanying the rejected transaction.

The task of correcting the transactions is difficult because it requires a large amount of knowledge about transactions; it's tedious; there may be more than one error in a transaction; and the rules are very difficult to read and interpret. It is often the case that a single transaction is cycled many times between the accounting system and the correcting group.

The rules are very long and unstructured, so it is very difficult for a non-expert to understand the rules. Once more, the rules are purely in English, so are open to misinterpretation. See figure 1 for an example rule.

1.1. What Do We Want?

It is hoped that an automated system, encapsulating the knowledge from the rule book, could be used as an expert assistant to the people responsible for transaction error correction. Ideally, rejected transactions could be feed directly to the new system, which could display transactions as an on-line form, with erroneous fields highlighted. Each highlighted field would be accompanied by suggested corrections. The user could either accept the suggestions or enter a new value, whereupon the entire transaction would be re-checked immediately (without cycling through the accounting system again). When the transaction is deemed to be correct, it would be returned to the accounting system. Simple, obvious errors could even be automatically corrected and returned to the accounting system without human intervention.

The rules of the business world change rapidly, so it is important to have a knowledge representation that is easy to understand and update. Since minor updates to the knowledge base will be done relatively frequently, we would like a system simple enough for the domain experts to update without extensive training. Given the complexity of the domain, this can probably only be done by hierarchically

structuring the knowledge base -- something most current expert systems shells don't do well. We draw an analogy here between hacking and unstructured programming.

INVALID FEDERAL TAX CODE - INVALID FTAX CODE

DESCRIPTION

A SALES INVOICE LINE WHICH REQUIRES A FEDERAL TAX INDICATOR SHOULD BE CODED WITH ONE OF THE INDICATOR VALUES SPECIFIED BELOW:

(A) FOR SALES OF AVIATION AND MOTOR GASOLINES (MAJOR PRODUCT CODES 101, 121, 211, 215, 221, 222, 224, 225, 228, 231, 235):

(1) IF THE PROVINCE TO IS '09' THE FEDERAL TAX CODE MUST BE 6.

(2) IF THE PROVINCE TO IS '02' AND THE TRADE CLASS IS '025' THE FEDERAL TAX CODE MUST BE 6.

(3) OTHERWISE THE FEDERAL TAX CODE CAN BE ANY OF 0, 2,4,6,7.

(B) FOR SALES OF TURBINE AND DIESEL FUELS (MAJOR PRODUCT CODES 141,142,322, 323, 325, 326, 327, 328, 329, 361)

(1) IF THE PROVINCE TO IS '09' THE FEDERAL TAX CODE MUST BE 7.

(2) IF THE PROVINCE TO IS '02' AND THE TRADE CLASS IS '025' THE FEDERAL TAX CODE MUST BE 7.

(3) THE FEDERAL TAX CODE CAN BE 8 IF THE DOCUMENT DATE IS PRIOR TO SEPTEMBER 3.1985.

(4) IF THE PRODUCT CODE MAJOR IS 361, THE TAX CODE MUST BE 2 OR 7.

(5) OTHERWISE THE FEDERAL TAX CODE CAN BE ANY OF 0, 2, OR 7.

(C) FOR SALES OF HEATING OILS (MAJOR PROJECT CODES 301, 302, 304, 305, 306,321, 324, 362) THE FEDERAL TAX CODE MUST BE 7 OR 8. EXCEPTIONS:

(1) PRODUCT CODE 362300 SHOULD BE TAX CODE 2 OR 7.

(E) FOR DISPLAY ADVERTISING (PRODUCT CODES 426-996, 426-997 AND 426-998) THE FEDERAL TAX CODE MUST BE 7.

(F) FOR AIRPORT FEES (MAJOR PRODUCT CODE 030), THE FEDERAL TAX CODE CAN BE 0, 2, 4, 6, OR 7. THE FEDERAL TAX CODE CAN BE 8 IF THE DOCUMENT DATE IF PRIOR TO SEPTEMBER 3, 1985.

(G) ON NON-EXPORT CARDLOCK SALES, (TRANSACTION 5670, OR 5600 WITH TRANSPORTATION CODE '44') FOR PRODUCTS 4XX, 5XX, AND 9XX, THE FEDERAL TAX CODE MUST BE '0'.

(H) FOR ALL OTHER SALES THE FEDERAL TAX CODE CAN BE 0,7 OR 8.

Figure 1.: A business rule.

2. History

We first tried to encode the informal rules in a commercial expert system shell. While this could be done, it turned out that a single informal rule expanded to an incredibly large number of shell rules. Furthermore, the shell rules where very difficult to understand and to map back to the original informal rules. Both due to predicted size of the knowledge base, and the predicted difficulty of maintenance, this avenue was rejected.

The next approach was to use the Z specification language to try to get a better understanding of the rules and get an idea of what kind of tool could implement a solution. Z turned out to be extremely useful to experiment with different methods of implementation, but it could not, of course, give us an implementation. After many paper-and-pencil experiments, it became clear that the rules could, for the most part, be rewritten as a hierarchy of rules with exceptions. Thus we can say things like "if A then X unless B then Y", which means "if A is true then do X, but if B is also true then we should do Y and not

X". Although this has a straightforward interpretation in simple rules[1], it is extremely hard for a user to understand (and maintain) the rules when the exception hierarchy becomes large[2].

3. KRS as the Prototype Platform

Now that we were aware of what we needed, we happened upon KRS [Gaines, 1991d], a knowledge representation server which not only contained rules with exceptions but several other features that applied to the problem. KRS contains a powerful visual language that we believe would be useful for the domain experts to input the rules directly themselves with minimum training. It also deals with sets and "negative sets" nicely, which is useful in the domain. Furthermore KRS can be easily linked to a hypercard stack where we can easily implement an on-line form interface.

3.1. Visual Language

The visual language allows the user to create graphics to represent the knowledge domain. The graphs are amazingly easy to understand (with a little training), and should significantly ease the maintenance problem over textual representations. The visual language is translated to KSSn, where it can be used by the KRS server. There is also a visual language that can translate from KSSn to the visual language. It is important to note that the visual language is not stored as KSSn and then reconstructed, but is actually stored such that the image is reconstructed exactly as the user had created it. It is critical to let the user arrange and group the symbols in a way that make sense to the user, not the machine. Simple and powerful graphical tools are provided for moving, grouping, cutting-and-pasting, etc.

3.1.1. Graphical constructs

The graph is composed of nodes with connecting arcs. There are six different nodes: primitive concepts, concepts, individuals, rules, roles, and constraints. Every node is and icon labeled with its name. There are two different arcs: directed arcs[3] and undirected arcs. Most of the constructs will be briefly explained below. For a more detailed description see [Gaines, 1991a]

Primitive Concepts



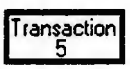
Primitive concepts are represented as a ovals with marks on either side. A primitive concept is a schema or class definition.

Concepts



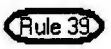
Non-primitive concepts are represented as ovals (without marks). A non-primitive concept is fully defined by its relationship to other concepts, a non-primitive is *recognized* when other entities comply with its definition.

Individuals



Individuals are represented as rectangles. An individual is an object with identity.

Rules



---

[1] "if A&~B then X, if A&B then Y"

[2] When there are multiple exceptions to A or when B, in turn, has exceptions.

[3] The directed arc is highly overloaded.

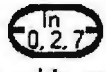
Rules are represented as rectangles with rounded corners. The input concept to the rule is the antecedent, and the output concept is the conclusion of the rule. In other words, if the input concept holds, then the output concept is asserted to hold. Multiple input arrows are considered to be disjunctive.

Roles



Roles are represented as the label text with no surround at all. Roles are used, in this case, to represent *attributes*, or *fields* of the transaction records.

Constraints



Constraints are represented as rectangles with rounded corners and marks at either side. Constraints are usually used with roles to type the role or for concept recognition.

Subsumption



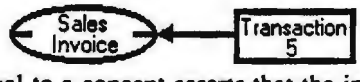
A directed arc between two concepts indicates that the concept at the tail is subsumed by the concept at the head. This is the isa relationship, or can be thought of as inheritance.

Disjointness



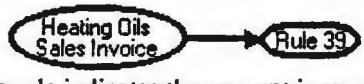
An undirected arc between two concepts indicates that nothing can be asserted as being a member of both concepts.

Instantiation



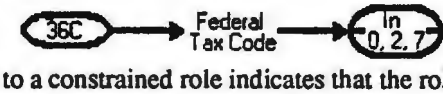
A directed arc from an individual to a concept asserts that the individual exhibits the properties of the concept. In other words, the individual is an object of the schema.

Antecedent



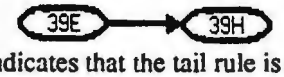
A directed arc from a concept to rule indicates the concept is an antecedent to the rule.

Conclusion



A directed line from a rule to a constrained role indicates that the role is constrained if the rule is fired.

Exceptions



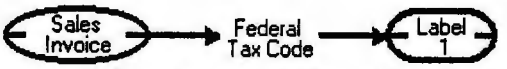
A directed arc between two rules indicates that the tail rule is an exception to the head rule.

Attributes



A directed arc from a primitive concept to a role indicates the role is an attribute of the concept (schema).

Types



When a role is attached to a primitive concept, a directed arc from the role to a constraint constrains the value that can fill that role, and so defines (or partially defines) type information about the role. In this case the role *federal tax code* is a label (string) with a cardinality of one.

Recognition



When a role is attached to a primitive concept, a directed arc from the role to a constraint is used for concept recognition. In this case, an *invoice* is recognized to be a *heating oils sales invoice* iff the *product code major* is 301, 302, 304, 305, 306, 321 or 324.

Multiple occurrences of the same node (same node type and label) in a graph are deemed to be the same entity, so the user may define a concept in one place and use it in another without problem. This enables grouping in complex constructs without an inordinate amount of arcs clogging the visual effect.

3.1.2. Comments

The user may place comments anywhere on the graphic surface just by creating a node (usually a role, since there is no surround) with no arcs connected to it. This is often very convenient for later maintenance when the rules have to be changed due changes in the business world.

3.1.3. Visual representation of a business rule

The visual language translation of the business rule given in figure 1 is shown in figure 2. This rule is easier to understand than the informal rule, and is not ambiguous. Another advantage is that the ambiguities of the informal rule have been removed. For example, when the *major produce code* is 141 and the *province* is '09' the informal rule can be interpreted as either allowing or disallowing *federal tax code* of 8 (see clauses B1 and B3); whereas the visual representation makes it clear that a federal tax code of 8 is not allowed (see lower left concepts *turbine & diesel fuels 09* and *new turbine & diesel fuels*).

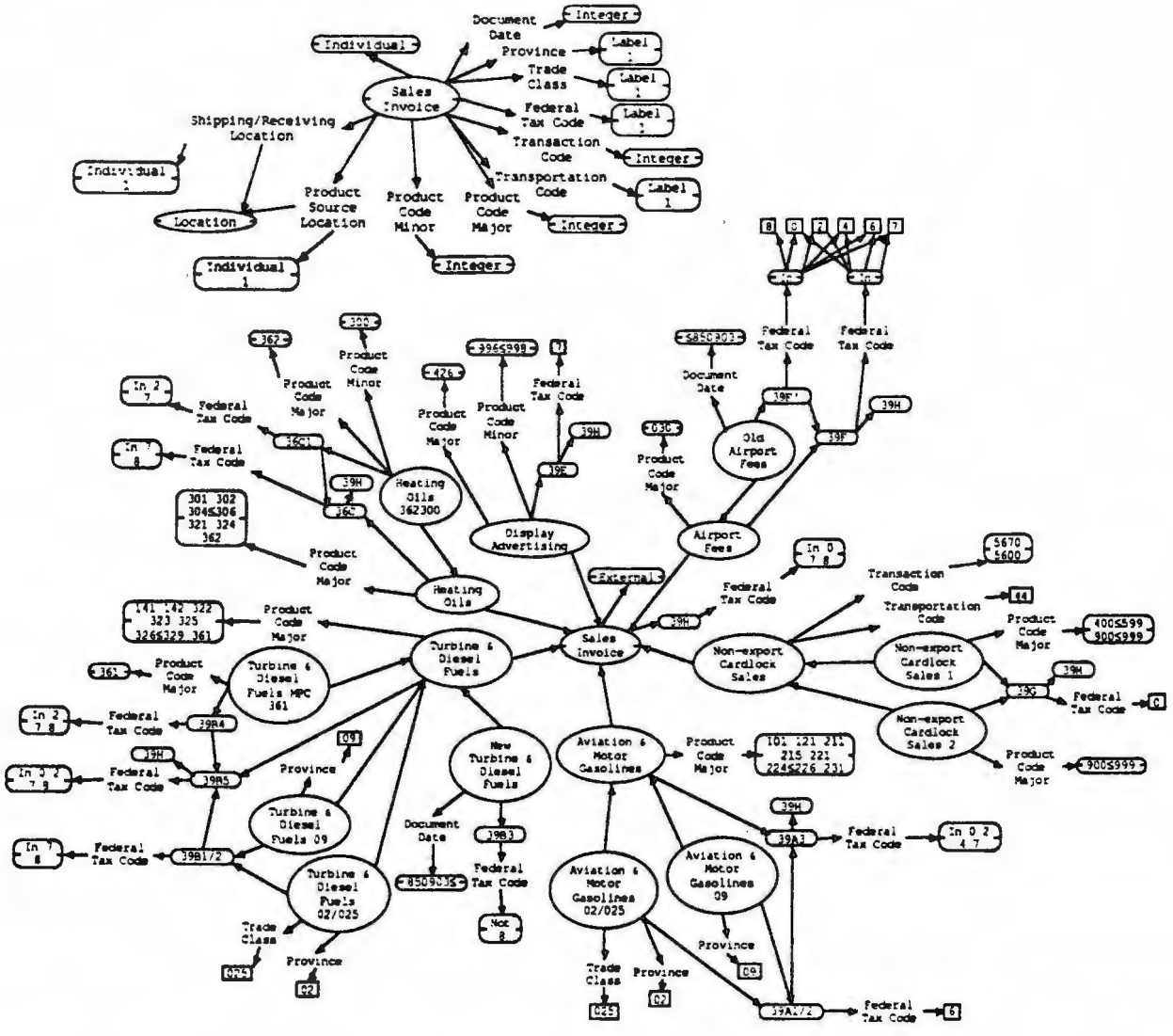


figure 2: Visual language representation of the rule in figure 1

3.2. Hypercard Link

A hypercard stack can communicate with a KRS process by opening a IPC channel to it and passing messages through the channel. Hypercard can send any message that is legal KSSn syntax. Likewise, return messages are the text expected from normal KSSn text interface. The hypercard stack must parse the return messages appropriately, but this is not difficult since the KSSn output has been designed to be easy to parse.

Hypercard is used for an on-line forms interface (see figure 3). In the case of the prototype, the user can fill in the form[4] and then click on the Check Case button to have the KRS run on the record. Any erroneous fields will be highlighted and a suggestions field opened to the right of the field. The user is free to make corrections and recheck the record.

[4] In the actual system, the form would be pre-filled from the accounting system.

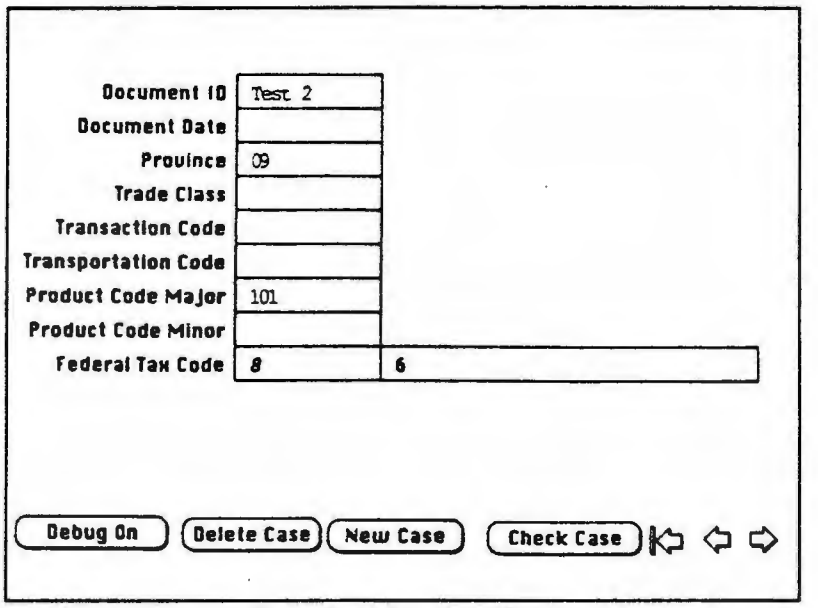


figure 3: the hypercard form

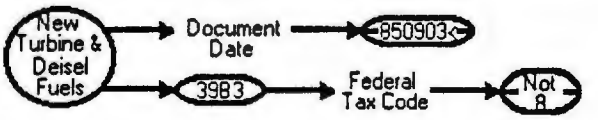
3.3. Sets and Ranges

The example informal rule points out the need for handling of sets with a set notation that can specify both discrete values and ranges of values. For example, almost every major (lettered) clause in the rule refers to the *major product code* field, and a list of ten or so discrete values that it may contain for the clause to apply. Clause G refers to *major product codes* in the 400's 500's and 900's -- awkward to list as discrete values. The former clause (for clause A) can be represented with a constraint with the label "In 101, 121, 211, ... 235", while the latter (clause G) can be represented with a constraint with the label "In 400≤599, 900≤999". The discrete notation and range notation may be mixed freely in a single constraint.

3.4. Negative Sets and Ranges

Another useful feature of KRS is its handling of negative sets. Clause B3 of the example rule is an example use. Clause B3 says that any transactions with *document dates* prior to a certain date are allowed to have *federal tax codes* of 8. Without negative sets, this clause would not have its own identity, but would result in two versions of the other clauses, one each for before and after the date.[5] This lack of direct correspondence makes for both an encoding problem for the novice user and a maintenance problem later on.

With negative sets, we can encode clause B3 by recognizing newer-than-the-date transactions, and adding a rule that newer transactions cannot have a federal tax code with value 8:



None of the other clauses are affected, and even if we add new clauses the date rule still applies to them.

3.5. Constraints

[5] For trivial example, see the visual translation of clause F (airport fees) in the upper right corner of figure 3. For a more telling example, try to encode clause B (turtine and diesel fuels) without using the **not** operator.

Constraints are used both with concepts for recognition and in conclusions of rules to constrain the values the roles may take on. The difference here is not superficial. Concepts constraints can be seen as *passive* constraints where we assume the data (value in the role) is correct and we are merely recognizing the concept. Rule constraints are *active* constraints where we make no assumptions about the correctness of the data but will enforce it. In the former case the data is driving the processing of the knowledge base; in the latter case the knowledge base is processing the data.

4. The Future: An Implementation Platform

KRS is currently implemented on the Mac, and the company demands that PCs, VMS Vaxes, and IBM mainframes be used exclusively. So KRS, in its present form, cannot be used as an implementation platform. If further work on the project proves successful, some appropriate subset of KRS could be translated to one of the other hardware platforms. Since KRS is implemented in C++, this should not be too onerous a task.

5. Conclusion

More work needs to be done to prove that KRS can represent all facets of this problem (the rule book looks like a telephone book), but I am extremely optimistic that it can be done. The final implementation will require a few add-ins like links to remote databases.

KRS has shown itself to be a very useful tool for developing knowledge bases for domains that are based on reasoning with exceptions. KRS's visual language has shown itself to be surprisingly useful for knowledge base development and maintenance. The visual language's most valuable attribute is enabling the user to give hierarchical structure and informal grouping (a problem with a lot of current expert system shells) to the knowledge base by allowing knowledge bases to be spread over multiple files and user placement of icons.

References

[Gaines, 1991?a] Gaines, B.R. "An Interactive Visual Language for Term Subsumption languages." ???

[Gaines, 1991b] Gaines, B.R. "Empirical Investigation of Knowledge Representation Servers: Design Issues and Application Experience with KRS." ???

[Gaines, 1991?c] Gaines, B.R. "Integrating Rules in Term Subsumption Knowledge Representation Servers." ???

[Gaines, 1991?d] Gaines, B.R. "Organizational Modeling and Problem Solving using an Object-Oriented Knowledge Representation Server and Visual Language." ???

CREATIVE KNOWLEDGE ACQUISITION: AN ANALYSIS

Timothy C. Lethbridge

Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
(613) 564-8155 tcl@csi.uottawa.ca

ABSTRACT

We discuss the idea of 'creative' knowledge acquisition (KA) by which we mean generation of primarily new knowledge using a knowledge-based system. We present a conceptual analysis outlining the properties of creative KA tasks. Then we develop a process model of creative KA based on a knowledge enterer's concept-elucidation objectives. We use the conceptual analysis and the process model to justify some recommendations relating to the knowledge representation and user interface of a creative KA system. The recommendations involve helping users generate, record and effectively sequence their thought processes. We are implementing many of the recommendations in version 4 of our CODE system.

INTRODUCTION

When surveying use of our lab's knowledge acquisition (KA) system, CODE (Skuce et al, 1989, Skuce 1991a), over the past few years, one major point in particular strikes us: CODE's use as a *creative assistant* has often been more valuable than the knowledge bases we have built/\.

For example, during a major industrial application of CODE to software requirements analysis, it took us many months to establish a substantial knowledge base and to convince project team members to use it as the central conceptual repository and documentation source (Skuce 1991b). Almost from the beginning of our involvement however, team members were able to benefit from the process of building the knowledge base: By the manipulation of conceptual structures through an effective user interface, complex issues with which they had previously struggled were resolved.

We have taken to creating very small knowledge bases with CODE to develop ideas in our research (including for this paper). The knowledge bases frequently become obsolete as ideas rapidly evolve; but that does not matter: the value has come from use of a KA system as an analysis tool to get us over the creativity 'hump'. In one domain, the design of our core knowledge representation, we have created knowledge bases from scratch three separate times in order to make use of the creative potential of the process.

The focus of this paper is the problem of how to optimize the creative potential of KA systems, whether the resultant knowledge base is to be maintained or not. We start by analysing the creative aspects of KA so we can understand what needs optimizing. Then we develop a hypothetical model to describe the process a user goes through when using a KA system creatively. Finally we use an example user session to make some suggestions about creative KA. These suggestions affect 1) knowledge representation, 2) ontology and 3) user interface.

1. CREATIVE KNOWLEDGE ACQUISITION

The general meaning we give to 'knowledge acquisition' (KA) is any process leading to the addition or refinement of knowledge (i.e. concepts) in a knowledge base. We prefer this broader meaning to that restricted primarily to 'expert systems' or 'expertise' (Gaines 1989); we do not however consider a task as common as writing a document to be KA, unless it is part of a process leading to knowledge base building.

By *creative* knowledge acquisition we mean KA of primarily *new* knowledge, due to direct manipulation of and feedback from conceptual structures evolving in parallel in minds and in a knowledge base. It is a largely individual process, comparable to creative writing, but with much more potential for tool support than possible with natural language composition.

By *new* knowledge, we mean knowledge not present in the mind of the domain expert or in the knowledge base before the KA session (although less refined knowledge may have been present). New knowledge is originated (i.e. created) in mind or machine using processes (deductive, inductive, intuitive etc.) that detect distinctions or patterns in pre-existing knowledge. Processes that are not considered to be originating knowledge include observing, translating, loading, decompressing, inheriting and combining (as long as these processes lead to nothing distinctive being observed about the knowledge).

We consider knowledge to be composed of *concepts* which are mental or mechanical representations of anything that can be 'thought' about. Two important types of concepts are independent concepts and the relations between them that we call *property occurrences*.

The early stages of creative KA can typically be described as 'individual brainstorming into a knowledge base', although brainstorming has a connotation of haphazardness that should not be applied to all creative KA.

Two particular aspects of KA-related literature are relevant to creative KA. One is the study of serendipity, the discovery of knowledge one is not looking for (Roberts, 1989). Creative KA involves more than just enhancing chances for serendipity: One *often is* looking for specific knowledge one does not possess. The other useful aspect of the literature is the use of KA in design (Reich, 1990; Mark, 1990); designers have much to benefit from the enhancement of creative potential.

2. PROPERTIES OF KA TASKS: COMPARING GENERAL AND 'CREATIVE' KA

Before we describe our process model for creative KA, we present a conceptual analysis comparing creative KA to general KA. The analysis is based on classifying KA tasks with the objective of identifying where we might target technique improvements. Other KA classification literature tends to be organized by subtask, technique, tool, application area, assessment criterion etc. (e.g. Wetter and Woodward 1990)

Our analysis is in the following ten subsections, each describing a semi-independent property on whose scale we can place any KA task. Following each property we indicate in what way creative KA tends to be specialized. Our model, described later, is developed with these specializations in mind, but may be applicable in more general KA situations.

The properties are listed in order, such that those that most distinguish creative KA are placed first. The properties are phrased as questions to be posed about each task; under each question is a list of points outlining a rough scale in which the answer should fall. The points listed for each property are not meant to be the only possibilities, nor are they meant to be mutually exclusive or necessarily form a logical progression. Within a given KA task, a property typically involves a range of points on its scale, which may apply at once or at

different times; this implies that some aspects of a task may be considered creative and some may not.

Property 1. At what level of clarity does the knowledge exist in the minds of the domain experts, prior to knowledge acquisition?

a) The knowledge is considered concrete and factual: The expert(s) know (have conscious access to) what they know. The KA process merely transfers the knowledge between human and machine.

b) The knowledge is is fuzzy, unsystemized etc: The experts may know what they know, but generally not precisely, and they generally cannot clearly characterize the knowledge which may be undeveloped 'ideas'. Typically, mental heuristic knowledge is at this level of clarity. The KA process crystalizes the knowledge.

c) The knowledge is latent, experiential etc: The expert(s) do not know what they know. The KA process draws out the knowledge.

d) The knowledge does not exist. The KA process generates new ideas by providing a medium for the organization of thoughts.

Answer d most strongly characterizes creative KA, although answer b also frequently applies. The knowledge is already 'created' in form a, and form c requires emphasis on elicitation techniques that detract from spontaneous creativity (see property 4 below).

Property 2. How is the knowledge originated?

a) Most of the knowledge originates as the result of computational processes (e.g. it is extracted from raw data using machine learning techniques).

b) Most of the knowledge originates as the result of human thought processes.

Creative KA falls at the latter end of this spectrum; creation of the knowledge is done non-algorithmically by human thought processes (frequently intuitive, and generally poorly understood by today's science).

Property 3. What formulation[1] of the knowledge is performed prior to its entry into the KB?

a) The knowledge is originally formulated as natural language documents, typically without the intent to create a KB.

b) The knowledge is gathered from the expert(s) in one form (natural language notes, diagrams, results of elicitation tools etc) and is reformulated when entered into the KB. The gathering from experts is done with the intent of adding knowledge to a KB.

c) None, the knowledge is elicited, formulated and stored as an indivisible interactive process.

Creative KA follows the pattern of point c. The idea is that feedback from the interactive process stimulates creativity.

---

1 By *formulating* knowledge, we mean representing it in some medium.

Property 4. What are the primary elicitation techniques used?

a) None. The knowledge is entered into the KB, with reports of errors, ambiguity and inconsistency as the only feedback. We do not consider such feedback alone to be an elicitation technique, although it can be made to trigger concept structuring (d below).

b) Discussion techniques (interviews, case studies etc.) with little machine involvement.

c) Concept structuring techniques (e.g. repertory grid, interpretive structural modelling) performed without machine assistance.

d) Exemplar-based techniques, usually performed with machine assistance.

e) Concept structuring techniques performed with machine assistance. The machine guides much of the KA process.

f) Agenda maintenance techniques: The machine attempts to help the user decide what to tackle next.

g) Alternate perspectives feedback: The user guides the KA process, receiving elicitatory stimulus by looking at alternate perspectives on knowledge already entered. The user's natural ability to make inferences in ways that could not be forseen by the machine is reinforced.

Although any machine assisted technique can assist creative KA, technique g, where the user is most in control, best characterizes our vision of creative KA.

There is extensive discussion of elicitation techniques in the literature. A decomposition comparable to the above may be found in (Gaines, 1990, p 9-19). A good survey of those applicable to expert systems is (Neale, 1989). Interpretive structural modelling is discussed in (Warfield, 1976).

Property 5. Who performs the formulation of the knowledge into the machine knowledge representation?

a) The machine itself, largely automatically from data, natural language etc.

b) The knowledge engineer acting as an intermediary who works with domain expert(s) away from the machine and then goes to the machine to enter knowledge.

c) The knowledge engineer acting as a facilitator who works with domain expert(s) in front of the machine.

d) The knowledge engineer, after having developed domain expertise: The true domain expert(s) act as critics.

e) The domain expert(s), with the knowledge engineer acting as active critic. The KE is present, guiding the process, but the domain expert(s) have control over the machine interaction. Semi-intelligent software may substitute for the KE.

f) The domain expert(s) on their own, with a knowledge engineer (or software) performing occasional critiquing.

g) The domain expert(s) acting as their own knowledge engineers.

Creative KA leans towards methods f and g, although all of c through g are possible. In one project we had experiences with methods b through f, and our system demonstrated its creativity-stimulating potential to the greatest extent using method d. We believe, however that this was an artifact of our having much greater tool expertise than the domain experts

did when they used methods e and f. A major thrust of our research is to increase the usability of creative KA systems in order to make methods f and g the norm.

Property 6. What type of expert collaboration is involved?

a) The process is characterized by group brainstorming.

b) The process involves parallel formulation of detailed alternate perspectives on the 'same' knowledge, followed by 'merging' and conflict resolution. This differs from brainstorming in that detail is generated prior to resolution.

c) The process alternates individual formulation of knowledge in areas where each expert has greatest expertise, followed by negotiation to resolve conflicts and disagreements.

d) The process involves individual responsibility for areas of the knowledge base with minimal overlap.

e) The knowledge source is primarily one individual.

We have used all methods except b, and although method a tends to stimulate the most creativity, little refinement of the knowledge is possible. Our concept of creative knowledge acquisition involves only the individual-work phases of methods b to e.

Property 7. What is the primary purpose of performing the knowledge acquisition?

a) To provide input to a program for use in its reasoning processes. This is the classical expert systems purpose.

b) To convey the knowledge to other humans: E.g. for a tutorial, documentation or retrieval system that performs little reasoning.

c) To define terms. E.g. when developing a domain's terminology.

d) To analyse, or better understand a concept; to explore thoughts. The knowledge base may be only temporary, or kept as a record of thinking (the secondary purpose is often b or e).

e) To design something: The knowledge forms requirements and/or specifications for the thing. It later becomes documentation about the thing and/or is actually used in processes that create the thing.

Creative KA can have any of these purposes, but those later in the list are more typical: Creative knowledge is gathered more for *immediate* problem-solving than with the delayed objective of solving problems *with* the knowledge. The process of representing knowledge tends to be more important than the knowledge. The addition to knowledge in people's minds tends to be more important than the addition of knowledge to the KB.

Property 8. What is the scale of the knowledge base?

Typically creative KA is performed on small discrete 'problems' (property 7, above) and created by individuals (property 6) hence knowledge base size is naturally limited. KB sizes have been limited by technology, but many people believe in a 'bigger (within limits) is better' philosophy, with long-term projects that attempt to capture much of the detail in a domain. While we laud such efforts, we believe in the need to develop special technologies for 'personal KB's. For a somewhat imperfect analogue, consider the differences between typical use of a large database system and typical use of a spreadsheet program.

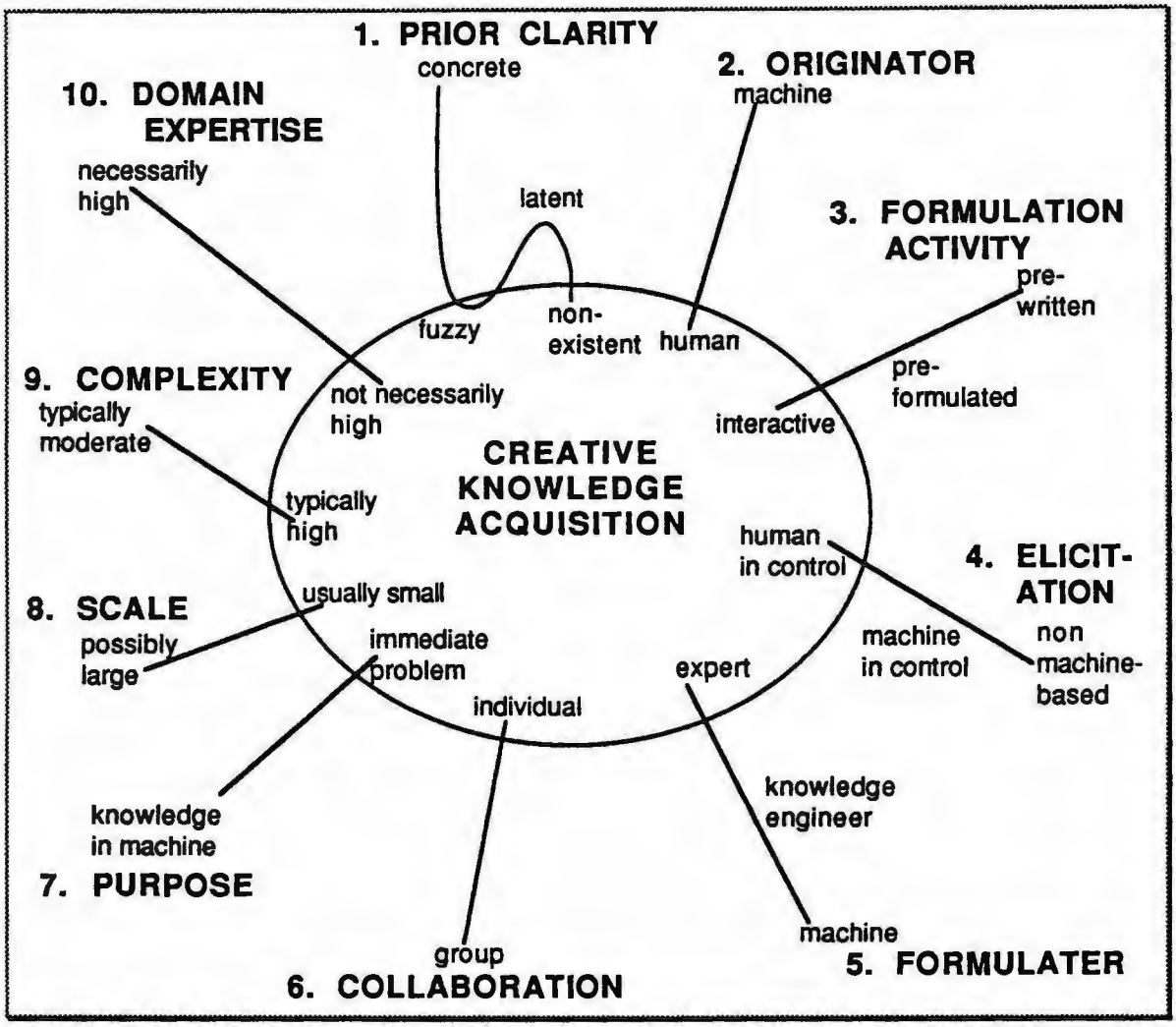


Figure 1: Semi-independent properties of knowledge acquisition tasks, showing how we characterize 'creative' KA. Each property is represented by a scale; properties of creative KA tasks tend to be specialized with values near the ends of the scales that are in the central ellipse.

Property 9. How complex is the knowledge acquired?

Complexity measures such things as interconnectedness, non-linearity, dimensionality, degree of meta-representation, degree of projection etc.

While smaller in scope, we see creative KA as tending to deal with greater knowledge complexity than mainstream KA. Complex problems are what people need creative KA for. Only in small KA efforts can time be afforded to expand concepts to extreme levels of detail.

Property 10. What is the level of expertise of the domain 'experts'?

While other types of serious KA require someone who truly knows the concepts (or has recorded them in documents), creative KA may involve non-expert knowledge source persons. Such persons may have peripheral knowledge and be using KA to expand their expertise (e.g. developing theories). Alternatively, such persons might be using KA to formulate 'naive' suggestions to real experts (in the same manner that someone might design their own house).

3. A PROCESS MODEL OF CREATIVE KA

In this section we describe our hypothetical model of the creative KA process. The model has two related components: 1) a model of knowledge-base editing objectives that may shed some light on how to assist a user in sequencing a creative KA task, and 2) a model of informal knowledge that can help explain how to prevent the user from getting bogged down too early in details.

We have developed the model in order to explain observed phenomena and to provide a framework for ideas about the improvement of creative KA. We make no claims about the cognitive correctness of our model; others may wish to persue such research. For us the model is primarily an aid to thought that we have found useful. In section 4 we will present several recommendations, several of which arise from the model.

Our model is not a 'user model' in the standard sense of the term. There is a literature on 'user models' and 'information source models' (e.g. Thost, 1989). We primarily differ in that we are not dealing with 'machine conceptualizations of particular users'; we do not emphasize automatically adapting interactions to such models. Instead we have a single general model that we use 1) to structure our own analysis, and 2) to suggest possibilities for facilities the user could call on, under his or her control, to aid memory and planning of a creative KA task.

KA as achieving mental 'concept elucidation' objectives

We model the mental state of a creative KA system's user as a loose collection of *concept-elucidation objectives*. Concept elucidation can be thought of as clarifying a concept, expanding the level of knowledge associated with it, building a conceptual substructure centred around it, or 'formalizing' it (see next subsection). Experience leads us to believe that creative productivity can be enhanced if the user arranges the objectives in a hierarchy.

We consider an objective to be a mental entity; the process of its achievement is performed simultaneously in mind and KA system. On the KA system side, the user builds machine conceptual structures; in an ideal system these correspond to the mental conceptual structures being built at the same time. We envisage the machine representation process assisting the parallel mental process — maintaining the network of complex details of which the mind would otherwise lose track. In other words, we see a sort of mind-machine symbiosis.

The hypothesized collection of objectives develops in steps as follows: The user mentally searches for, and seizes on, an idea for a conceptual structure that, if represented, would help achieve a particular elucidation objective. Representing the idea becomes a sub-objective. The sub-objective, and possibly parent objectives, are removed if the representation of the idea's central concept satisfies them, i.e. if the user feels that no further elucidation is necessary, that the conceptual structure has been completely built.

Steps of the preceding form take place iteratively until there are no more objectives to achieve. The objective collection grows and shrinks, and the conceptual structures in mind and machine become elucidated.

At each step, just one concept is actively changed in mind and machine (although passive consequences caused by effects such as inheritance usually also occur). It is important not to confuse the objectives with the conceptual structure being built, even though in our model each step involves one objective and one central concept.

To summarize the above, at each step:

- An objective is added in the user's mind. This may be a sub-objective of any other objective. We call it the*current objective*; the sequence of current objectives over time corresponds to the user's stream of consciousness.
- A user interaction may be made to commence satisfaction of that objective on the machine side. This may be nothing more than recording an empty independent concept. Other possibilities include adding a property occurrence, deleting a concept etc.
- The objective and a chain of its parents may be removed. Removal of an objective implies it has been achieved by the user interaction. When the last objective is removed, the task is complete.

We do not intend by our idea of a loose 'objective hierarchy' to suggest that the minds of creative KA system users work in a Prolog-like manner. To avoid this perception, it has been suggested that a blackboard architecture might be a superior model. We agree that such an approach could be useful, but it ignores the creative advantage we ascribe to structuring creative thinking hierarchically. Further research may shed some light on whether our view is justified.

KA as the process of increasing concept formality

We describe the process of elucidating a concept in the machine as making it more *formal* (Lethbridge, 1991a). We use the word 'formal' with its general meaning of 'having shape or form'. A concept becomes more formal if more links are made from it to other concepts and/or if other concepts it is linked to become more formal. Two concepts are *linked* if a property occurrence of one refers to the other (directly or indirectly).

In our model, the user's decision that an objective is achieved is based on his or her perception that the objective's central concept has reached a desired level of formality. The desired level of formality should presumably be higher for concepts of a higher-level objective; in fact if a user elucidates peripheral concepts to an excessive level of formality, he or she can be said to be *digressing* (helping to prevent this is a separate research problem).

When many concepts are first represented in the machine, they are *informal* since they lack links; they typically contain only descriptive text. We envisage the early phases of creative KA as involving the addition of large numbers of informal concepts very quickly (one step every few seconds). In later phases we see greater emphasis on increasing formality as objectives become more nearly achieved. In many applications, we see KA eventually moving out of a creative phase into one of validation and debugging — at such a time we envisage use of a somewhat different set of KA tool facilities from those used to enhance creativity. We also envisage a KA task moving from creative mode to validation mode and back again, several times.

How can we use the model?

There are three primary aspects of the creative KA process where our model gives rise to ideas for machine KA assistance: 1) Helping the user generate the initial idea that forms an objective, 2) Allowing the user to record his or her stream of consciousness without impedance, 3) Helping the user to organize his or her work:

- Generating the idea: We can provide initial knowledge, feedback and ways of viewing the knowledge.
- Recording the stream of consciousness: We must allow the user a) to follow his or her preferred path without distraction, and b) to represent anything that comes to mind as quickly and elegantly as possible.
- Organizing work: We can help the user to backtrack and to remember what her or his objectives are.

In section 4 we will outline some recommendations that arise from these points, but first we present a slightly more detailed look at the third point.

Organizing the creative KA task

We can attempt to keep a machine model of the user's elucidation objectives by maintaining a history of the corresponding concepts the user edits. By comparing the relationships of concepts in this history we can arrange it in a hierarchy ordered by sub-objective. Various aspects of the user interface can manipulate this hierarchy to assist the user:

- Sequencing assistance: We belive that helping the user to progress in a logical sequence may enhance creative potential. A logical sequence would consist of always progressing by creating a sub-objective of the current objective or by backtracking in the objective hierarchy when the desired level of formality is reached. A visual display of the objective hierarchy could help keep the user on track.
- Undoing and exploration assistance: The backtracking mentioned in the previous point was non-destructive; after completing work, the user steps back to the most recent incomplete objective. An alternate use of the objective hierarchy would be destructive backtracking, i.e. an extension of the standard 'undo' UI technique.
- Incompleteness detection: After extensive knowledge base editing, it can be useful to review leaves of the objective hierarchy, to see if there are any possible incompletenesses.

Although we have not yet applied these techniques to knowledge base editing, preliminary experience with some of these ideas in a CASE environment (an enhanced Smalltalk browser) has convinced us of their usefulness.

Generality of the model

Although we described the above model using the perspective of a knowledge source person working in intimate comunion with a creative KA system, we do not discount other possibilities. For example there could be a knowledge engineer involved (property 5), or there could be several knowledge sources (property 6). Such possibilities do not invalidate our model, they merely complicate it. For example, the objective hierarchy would be the abstract 'consensus' objective hierarchy (the idea of an objective hierarchy is abstract anyway, so making it more so is not of great concern!)

4. AN EXAMPLE USING THE PROCESS MODEL

In this section we step through an example creative KA session, making suggestions about how to prevent various creativity problems. Figure 2 (which is split into several parts) shows the session's progress.

We assume the presence of a basic knowledge representation (KR) system with a frame-like knowledge structure and multiple inheritance. Our suggestions describe facilities needed above and beyond this. Some of our suggestions are well-known, others less so.

The suggestions relate to three areas of KA system requirements: 1) knowledge representation, 2) ontology and 3) user interface. Although these tend to be researched separately, we believe it is important to consider them together for the purpose of solving problems such as creativity enhancement.

The list of recommendations is *not* intended to be comprehensive. In particular, it is not at all concerned with the following problems we believe are orthogonal to enhancing creativity.

1) Getting the knowledge right (making it consistent, correct, unambiguous or complete).
2) Ensuring the knowledge gathered is useful, i.e. preventing digression or excessive detail.
3) Following standards of knowledge representation style, e.g. to help keep knowledge bases consistent and comparable.

Some of our recommendations *may* help solve these problems, but some might in fact hinder such solution. For example, we discourage the user from stopping to worry about the consequences of entering a piece of knowledge in the early stages of KA. Such stopping may help get the knowledge right and ensure it follows standards, but it can hinder creativity.

We have made the observations described in this section repeatedly in a number of small KA projects. The particular example is synthetic and extremely over-simplified, but the domain is that of a real project.

Our research work is ongoing, so we expect to further refine the ideas in the future. Some of the recommendations use the terminology and conceptual framework of CODE, but we believe they can readily be transferred to the context of any other system. For greater detail on some issues see (Lethbridge, 1991b). For most of the recommendations, there exist compliant KA systems; we believe however, that no system available today complies with all recommendations.

Setting the scene

Imagine the following: A designer has in mind a new type of telephone service. He or she mentally develops the idea, but then realizes it has many complexities that are hard to organize in natural language, diagrams or other conventional media. He or she also wants to be able to explore options and receive automatic feedback about the idea. For these and other reasons it is decided to use a 'creative KA' tool.

Step a: Creating a blank slate

The user starts with a top-level objective

Suggestion a1: Multiple interaction paradigms should be available, including graphical and textual; but where possible the paradigms should share homologous appearance, operations, feedback modes etc.

In figure 2a, the user is using a graphical view (see Joseph, 1989) that operates like a drawing program.

Textual views organized alphabetically or hierarchically should also be provided. These might operate like an outline processor The idea is to give the user maximum flexibility to record thoughts.

Suggestion a2: It should be possible to construct various types of 'maps' of parts of the knowledge base.

In figure 2a, the user is operating on an empty subset of the KB where any independent concept or property (relation) may be explicitly displayed.

In order to cater to the user's thought patterns, the user should be able to easily request display of any pattern of concepts and relations. We call such patterns 'knowledge maps' (Lethbridge, 1991b). Techniques for specifying knowledge maps include selection through browser or 'hyper' links, and arbitrary logical expressions to select or hide knowledge. Types of knowledge map include standard isa hierachies, property hierachies (suggestion h1), hierarchies of arbitrary transitive relations (e.g. 'part-of'), semantic networks, and many others.

Suggestion a3: The user should not be forced to name a concept, in fact the user should not be forced to provide any information he or she is not ready to provide.

In figure 2a, the name has not yet been provided for the new concept.

Naming is particularly distractive, requiring thought processes entirely different from those of concept creation. The user has to either choose the correct term from those that exist or create a neologism; either process can be difficult and the result may be unnecessary naming of an abstract subtlety. Hurriedly concocted names can 'stick' and cause confusion later. Some systems unfortunately require concept naming for 'internal' identification purposes. Position in a window, position in conceptual context and/or an automatically generated label or icon should however allow the user to distinguish the concept from others.

Suggestion a4: Maintain separate concepts for different levels of representation.

In figure 2a, a separate metaconcept *might be created to represent the description of the new empty concept.*

Such a concept would maintain meta-knowledge properties such as 'who did it', 'orientation on graph', etc. This separation of concerns can help keep the conceptual structure clear in the user's mind, but more important from a creativity standpoint is encouraging the proper application of *metarepresentation* (keeping the prepresentation of a representation of x separate from the representation of x). We have found poor understanding of this idea to be one of the greatest sources of hidden difficulty when creating a knowledge base.

This is one of a series of suggestions we will make regarding the importance to creativity of allowing the user to handle subtleties of representation. In general, the user should be able to readily distinguish between subtlely distinct things so as to be able to 'say different things' about them.

We believe there are too many subtle representation issues still to be researched for knowledge representation 'standards' to be developed yet.

Suggestion a5: Allow any level of formality of concept representation.

In figure 2a, the concept is completely informal except for invisible links to a metaconcept (suggestion a4).

See section 3 for a discussion of informality.

Step b: Naming a concept

The user has the lower-level objective to name the concept (despite suggestion a3 above, naming in this case is easy). Once naming is done, the naming objective is removed from the objective hierarchy. Naming is but one of a vast number of 'potential objectives' opened up by the top level objective, naming is merely the first that occurred to the user.

Suggestion b1: The interaction should be non-modal.

In figure 2b, the user should have been able to specify the name without requesting a prompt.

All basic interactions should be one-step processes so the user can change his or her mind, suspend an operation etc.

Suggestion b2: Knowledge manipulation, where possible, should be direct rather than performed using some command-based interface.

In figure 2b, the user should have been able to simply type over 'NNCF telephone'.

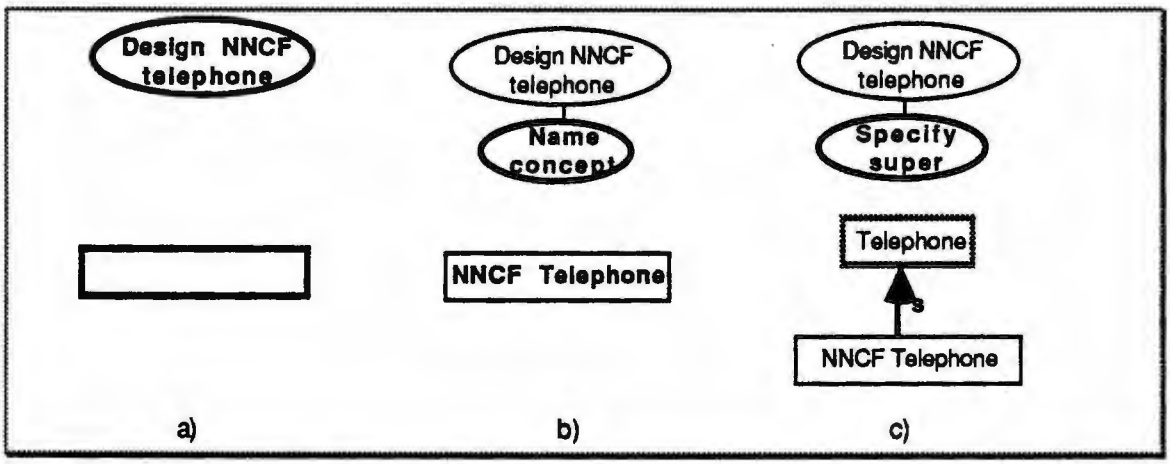


Figure 2 a,b,c (part 1 of 5): Each lettered section represents a user's step towards elucidating his or her idea for a telephone design. The ellipses represent a hierarchy of typically subconscious user objectives, the bold objective being current (the system may attempt to track this hierarchy, but it is not actually seen on the screen). Below the ellipses are conceptual graphs of the growing knowledge base as drawn by the user. Knowledge added to satisfy each objective is in bold (entirely new) or halftone (pre-existing knowledge merely added to this graph).

Step c: Specifying a superconcept

The user wants to say what the 'NNCF telephone' *is*. A superconcept link is established to the concept Telephone, which already exists in the knowledge base.

Suggestion c1: An ontology must be provided as a starting point for the KA task.

In figure 2c, the concept 'telephone' already existed in the ontology.

In the general case there must be sufficient concepts in the ontology to prevent the user from being forced to digress excessively (Skuce, 1990).

Suggestion c2: Effective ways to make references must be provided.

In figure 2c, the user should be able to locate 'Telephone' without excessive delay.

Making references can be done by a) locating concepts (via pointing, menus, wildcard patterns etc.) and/or b) composing a concept reference from other references (e.g. 'lawnmower of suburbanite'). It can be facilitated textually or graphically. Being able to refer to a recent history of references can also speed reference-making.

Step d: Preparing for a related concept

It has occurred to the user that there is a microprocessor involved somehow. The first step is to record a new concept.

Suggestion d1: The user should be able to open as many views into the knowledge as desired, in different windows. Furthermore, the user should be able to tie windows together to form browsers.

In figure 2d, the user may want to separate the microprocessor specification from that of the telephone itself. The user may want to explore existing knowledge about microprocessors before proceeding.

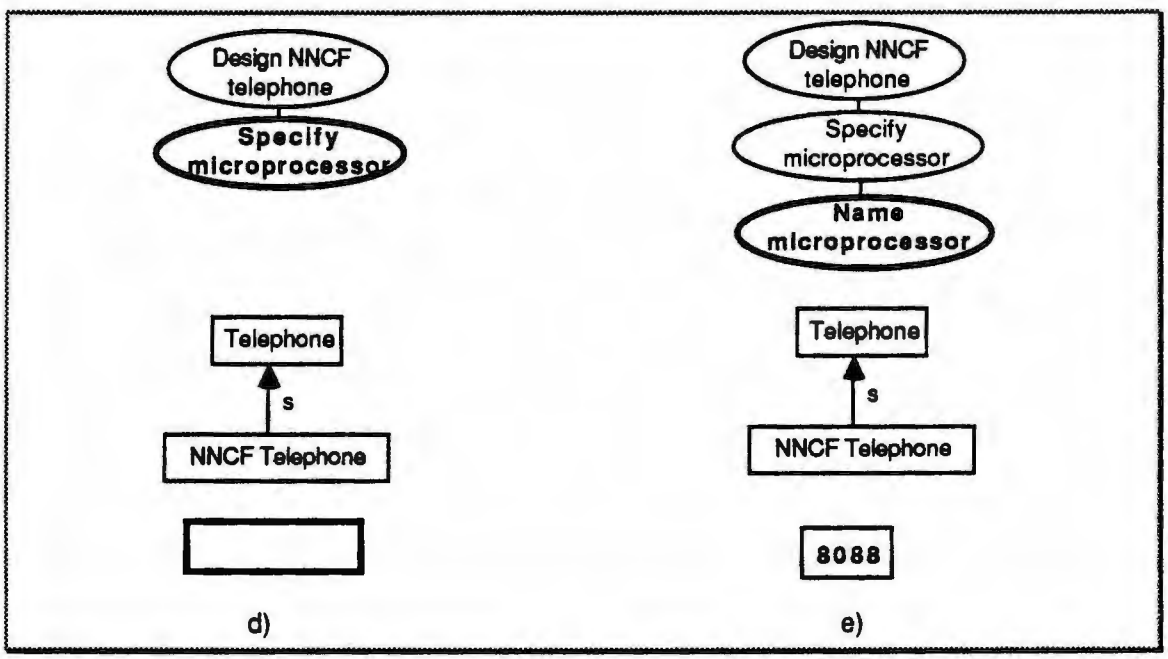


Figure 2 d & e

Step e: Naming the related concept

The user names the concept '8088'.

Suggestion e1: Concepts with the same name should be allowed.

In figure 2e, the user should not have been slowed down if another concept named '8088' had already existed.

Step f: Preparing for a property

The user has not finished specifying knowledge about the microprocessor, but his or her stream of consciousness is diverted into recording the fact that the telephone is to have a program. This is a sub-objective of the main objective.

Suggestion f1: The user should be able to specify links separately from their related concept.

In figure 2f, the user has specified a property occurrence rather than an independent concept to record his or her idea about 'has a program'.

The user should be able to work on a representation idea from either the relation or the independent concept perspective, whichever occurs to him or her.

Suggestion f2: Allow levels of formality when specifying properties.

In figure 2f, the new property has been specified very informally, with nothing more than a place-holder icon.

See also suggestion a5.

Suggestion f3. Provide assistance with tracking the objective hierarchy.

In figure 2f, the user has (for the first time) diverged from a strict stack of objectives, so objective tracking assistance might prove useful.

See section 3 for discussion.

Suggestion f4: Support should be provided for inverse properties.

In figure 2f, many properties of microprocessors and programs (e.g. memory size/requirement constraint, runs/runs-on) are mutual inverses.

The system should maintain such inverse relationships automatically. The main justification from a creativity point of view is prevention of distractive duplication of work.

Suggestion f5: Support should be provided for parallel properties (i.e. those whose refinement is similar in corresponding subtrees of an isa hierarchy).

In figure 2f, the 'purpose' of the microprocessor might be that of the program.

This is not a manifestation of inheritance, it is a form of concept-dependent constraint entered by the user. Often have we wasted time copying-and-pasting parallel properties. In general we do not follow Lenat's philosophy (Lenat, 1990) that such is a good way to enter knowledge: It reduces knowledge maintainability, and although it may cater to very short term creativity (recording stream of consciousness) in the medium term it impedes it by doubling work and increasing fear of inconsistency.

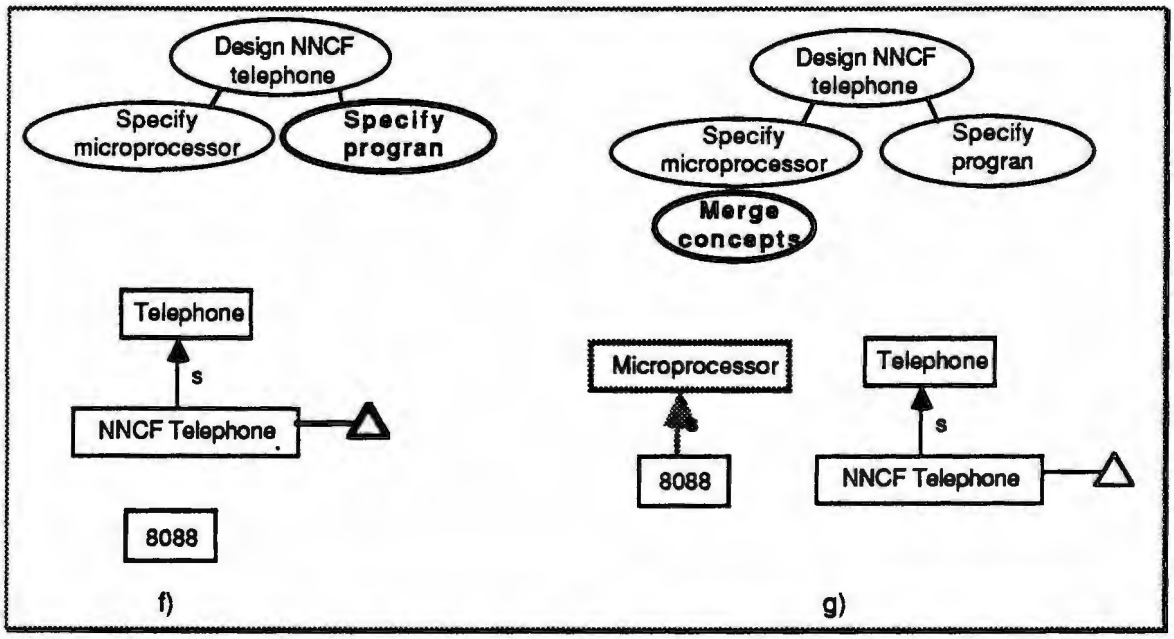


Figure 2 f & g. The triangle represents a 'property occurrence' (a conceptual relation)

Step g: Merging a concept

The user reverts to the 'Specify microprocessor' objective since the system has indicated that '8088' was already in this particular knowledge base. The operation performed is to merge concepts (unifying both them and references to them in the knowledge base; the result has the union of properties).

Suggestion g1: Provide sufficiently powerful editing primitives.

In figure 2g, the user performs a concept merge, one of many less frequent primitive knowledge-editing operations.

Aside from standard adds, deletes, renames etc., a number of less frequent primitive editing operations are needed to prevent side effects of operation composition. For example, it is necessary to ensure inheritance links are not temporarily broken (with the resulting loss of subtle knowledge (Lethbridge, 1991b)). An example of a less frequent editing primitive is 'concept reparenting'. All editing primitives need effective reference-making aids (suggestion c2)

Observaton g2: Help the user remember unfinished work.

In figure 2g, the user has not followed a logical sequence through the hierarchy, so she or he may later forget that about the objective of specifying the program.

Providing, on demand, a list of concepts visited but not recorded as 'satisfied' might be useful.

Step h: Adding a named property

As a further step towards specifying the microprocessor, the user indicates that NNCF telephones have electronic parts.

Suggestion h1: Organize properties hierarchically.

In figure 2h, the relation 'has electronic parts' is clearly a refinement of the relation 'has parts'. Whereas 'has parts' had previously inherited from some superconcept like 'device', the operation performed here makes 'has electronic parts' inherit as well (passively adding a superconcept 'device with electronic parts' to 'NNCF telephone' — not shown here)

We have found that having hierarchical properties (as opposed to the usual 1-level slot) substantially improves the power and compactness of the representation, and helps guide the user to the appropriate knowledge.

Suggestion h2: Provide a full ontology of properties.

In figure 2h, the user finds the 'electronic parts' relation to be already present, so creativity is speeded.

Not only do we emphasize that properties should be attached to the ontology of independent concepts, but the hierarchy of properties should be independently examined with reuse in mind.

Suggestion h3: Make property occurrences distinct from properties (i.e. ensure occurrences of conceptual relations and abstract conceptual relations are distinct concepts)

In figure 2b, while the property 'electronic parts' had previously existed, its occurrence in 'NNCF telephone' had not.

This is a another subtle representational distinction that we found essential to make (see suggestion a4). The set of things one can say about 'has parts' as a general relation is very different from what can be said about 'NNCF Telephones have electronic parts'.

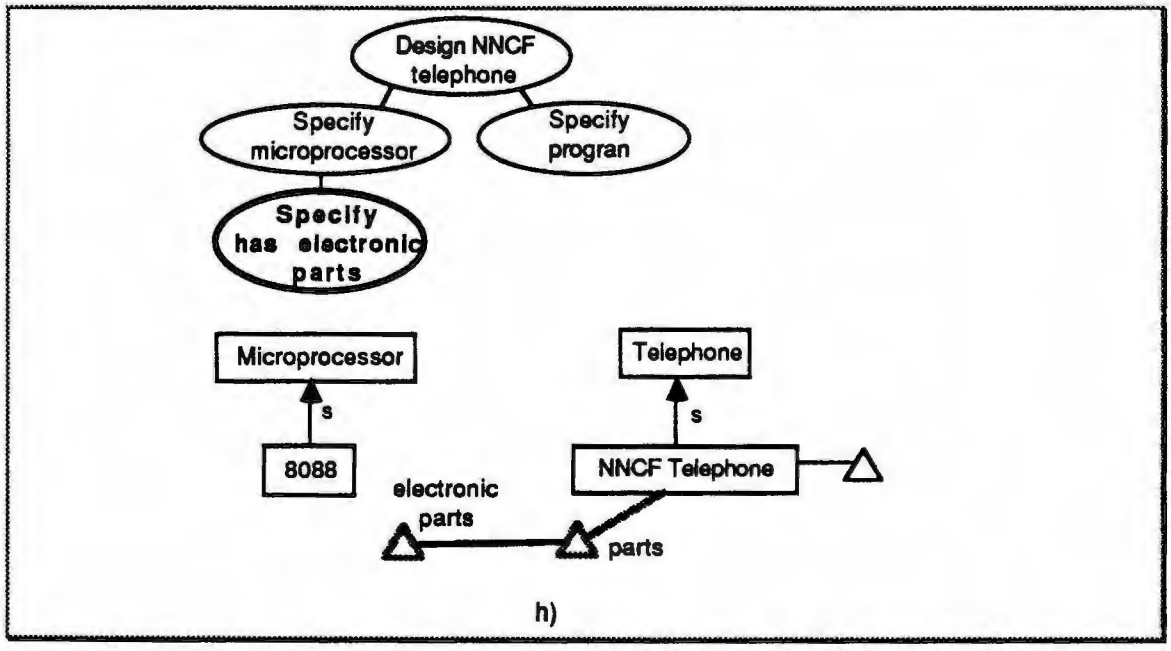


Figure 2 h. Properties such as 'has parts' are normally abbreviated (i.e. to 'parts')

Step i: Specifying an associated concept

The user indicates that the '[some] electronic parts of NNCF telephones are 8088s'. This many seem odd or incorrect; in fact it is only incomplete.

Suggestion i1: Keep associated concepts distinct from their superconcepts by default.

In figure 2i, the concepts '8088' and '8088 electronic part of NNCF telephone' should be distinct, because different properties apply to them. The latter, for example has the property 'contained in NNCF telephone'. This distinction is not shown visually in the figure.

This is another subtle distinction (like suggestion a4) that tends to be overlooked in most KR languages.

Suggestion i2: Feedback of suggestions made or problems detected should occur as soon as possible but not impede the user.

Figure 2i is the first step in the example where logical deduction performed by the KA system might detect problems. For example, '8088' might not be a valid refinement for the property 'electronic parts' as inherited.

Some kind of feedback should be provided immediately (a window listing problems, a colour change) but the user should have the option of responding to the feedback or not. As was pointed out in property 7, quite often the user may be less concerned with correctness and more with working out ideas.

Suggestion i3: Incompleteness should be tolerated.

In figure 2i, the user has not yet specified any other parts.

Unless asked, the system should not make deductions based on the assumption that the knowledge specified is complete, even for the purpose of passive feedback. To do so would be more distractive than worthwhile. (Inconsistency should also be toleraated, see suggestion s1).

Suggestion i4: Associated concepts should be specifiable in a dynamically interpretable way.

In figure 2i, we can assume the user typed 'an 8088' as the 'value' of 'electronic parts of NNCF Telephone' and that the system interpreted this as a request to create an associated subconcept of '8088'.

Another option the user may choose is typing some arbitrary natural language that the system cannot parse: In such a case, the user is recording knowledge 'informally' (see discussion in section 3). If the user had entered 'a widget', and the system had no concept of widgets, then the entry would be recorded as informal.

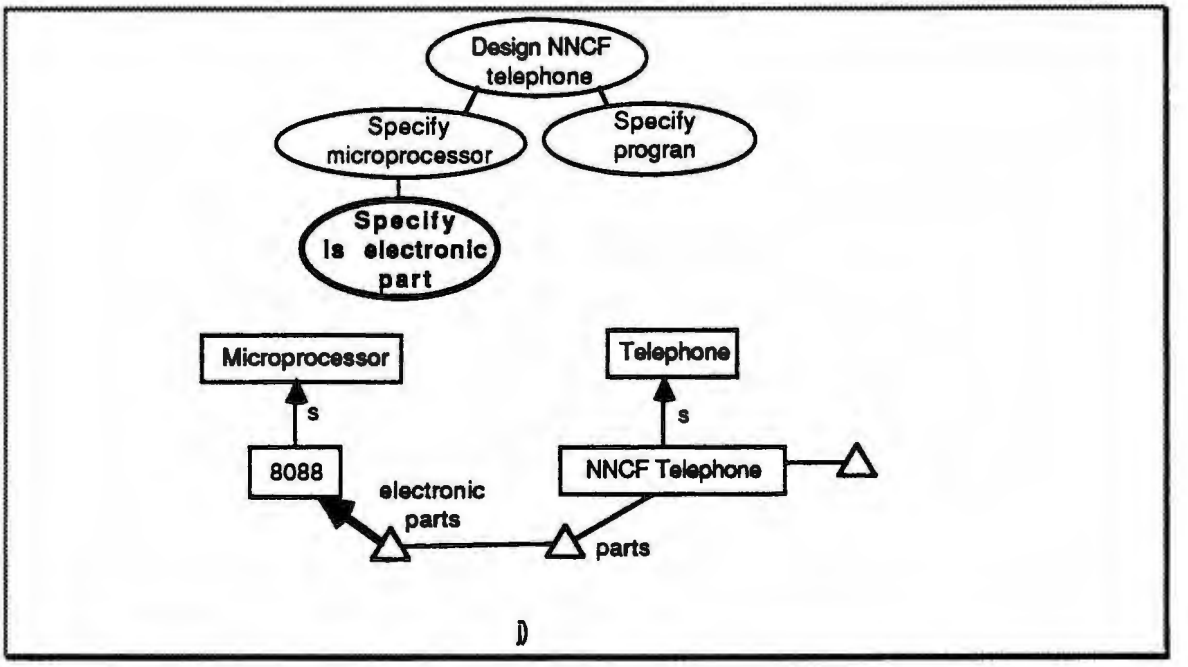


Figure 2 i

Subsequent steps

The nine steps described in this example should ideally take about a minute to enter (in a good system with an experienced user, we anticipate five seconds per step not counting thinking time). The user may continue to rapidly expand the knowledge base, eventually reaching a level of satisfaction that a sufficient level of detail has been recorded.

Suggestion s1: Commitment to the KB should be a separate step.

The user has to be confident that he or she is not 'damaging' existing knowledge, or entering 'unrepairable' knowledge. Three techniques can assist this: a simple undo capability, flagging changes as 'pending' until explicit commitment and providing a more complicated 'back-out' of committed changes. We believe that any kind of logical inconsistency should be tolerated up to the point of explicit committment (see suggestion i2). Commitment to the KB would imply that updated knowledge becomes visible in all views.

Suggestion s2: Powerful knowledge base restructuring techniques should be available so the user can correct errors and apply standards even after committing changes.

The justification for this is the same as in suggestion s1: Without confidence that problems can be fixed later, the user will slow down to get things right and may avoid entering concepts in which he or she lacks confidence.

Suggestion s3: Ensure that the system can be operated at a basic level with little or no training.

This may seem to be an elementary observation, when there are large numbers of primitive operations available (suggestion g1) it is important even to experts that basic operations can be performed without having to remember how. If a user has to look up how to perform an operation, his or her stream of consciousness will be interrupted.

5. CONCLUSIONS

Knowledge acquisition systems can be used in a largely creative mode where the actual knowledge represented in the machine may be of secondary importance to the inspirations created in the mind of the system's user. By analysing what contributes to creativity when performing KA, we have identified several key recommendations to enhance the creative potential of a KA system.

Our production KA system, CODE Version 2.4, follows some of the recommendations, and gave us many insights leading to the development of the other recommendations. At the time of writing we are implementing the latest version of CODE, version 4, which we anticipate will eventually fulfil all the recommendations.

A general recommendation for a creative KA system can be summarized as follows: "Facilitate the user's thinking - avoid inhibiting it". To be more precise, we present a list of recommendations below. Following each recommendation we indicate:

a) the suggestions from section 4 that the recommendation summarizes,

b) whether the recommendation applies to user interface (UI), ontology (ONT) or knowledge representation (KR), and

c) whether the recommendation is to help generate ideas (GEN), to help record the stream of consciousness (REC), or to organize the KA task (ORG).

List of recommendations

1. Give the user a sense of control over and immersion in the knowledge and system (suggestions a1, a2, b1, b2, f1, g1, i2) (UI, KR) (GEN, REC)
2. Don't force the user to follow particular thought paths (suggestions a3, a5, b1, e1, f2, i2, i3) (UI, KR) (REC)
3. Make allowances for subtleties of representation so that anything wanted can be represented in an intuitive way (suggestions a4, f4, f5, h1, h3, i1) (KR) (REC)
4. Provide the user with as much starting knowledge as possible (suggestions c1, h2) (ONT) (GEN)
5. Provide capabilities that prevent duplication of work (suggestions c1, c2, f4, f5, h1) (UI, ONT, KR) (REC)
6. Maximize the speed with which the user can make connections among concepts (suggestions c1, c2, h1) (UI, ONT, KR) (REC)
7. Make the tool adaptable to the user's preferred patterns of thought and work (suggestions a1, a2, d1, s3) (UI, KR) (GEN)
8. Prevent the formation of psychological barriers to the user's progress (suggestions s1, s2) (UI, KR) (REC)
9. Assist the user to track the task, forwards and backwards (suggestion f3) (UI, KR) (ORG)
10. Help the user remember unfinished work (suggestion g2) (UI, KR) (ORG)

ACKNOWLEDGEMENTS

Support for this research is or has-been provided by Bell-Northern Research and the Natural Sciences and Engineering Research Council of Canada. The author thanks Douglas Skuce, Ingrid Meyer, Yves Beauvillé and Bruce Porter for their ideas and inspiration.

REFERENCES

Gaines, B.R. (1989). Design Requirements for Knowledge Support Systems, *4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 12-1.

Gaines, B.R. and Shaw, M.L.G. (1990). Cognitive and Logical Foundations of Knowledge Acquisition, *5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 9-1.

Joseph, R.L. (1989) Graphical Knowledge Acquisition, *4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 18-1.

Lenat, D. and Guha, R.V. (1990). *Building Large Knowledge-based Systems*, Addison-Wesley.

Lethbridge, T.C. (1991a). A Model for Informality in Knowledge Representation and Acquisition,*Workshop on Informal Computing*, May 29-31, Santa Cruz CA.

Lethbridge, T.C. (1991b). *CODE Version 4 Reference Manual*, in preparation.

Mark, W. and Schlossberg, J. (1990). Interactive Acquisition of Design Decisions, *5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p 20-1.

Neale, I.M. (1989). First Generation Expert Systems: a Review of Knowledge Acquisition Methodologies, *The Knowledge Engineering Review*, pp. 105-145.

Reich, Y. (1990). Design Knowledge Acquisition: Task Analysis and a Partial Implementation, *5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 26-1.

Roberts, R.M. (1989). *Serendipity: Accidental Discoveries in Science*. New York, Wiley.

Skuce, D., Wang, S., and Beauvillé, Y. (1989). A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis, *Proc. 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 31-1.

Skuce, D., and Monarch, I. (1990). Ontological Issues in Knowledge Base Design: Some Problems and Suggestions, *Proc. 5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 32-1.

Skuce, D. (1991a). A Wide Spectrum Knowledge Management System, Submitted to: *Knowledge Acquisition*.

Skuce, D. (1991b). Knowledge Management: A Tool and Its Use in Software Engineering Design, Submitted to: Eighth IEEE Conf. on AI for Applications.

Thost, M. (1989). Generating Facts From Opinions With Information Source Models, Proc. IJCAI, Detroit, pp. 531-536.

Warfield, J.N. (1976). *Societal Systems: Planning, Policy and Complexity*, New York, Wiley.

Wetter, T. and Woodward, B. (1990). Towards a theoretical framework for knowledge acquisition, *5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff (Canada), p. 35-1.

Using KADS to Build K-ONCOCIN: A Conceptual Model of ONCOCIN

Marc Linster
Expert System Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1
FRG

Mark A. Musen
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University School of Medicine
Stanford, CA 94305-5479
USA

ABSTRACT

In recent years, builders of expert systems have become increasingly interested in the KADS (Knowledge Acquisition and Design Structuring) method of knowledge acquisition. Despite the apparent face validity of the KADS approach, reports of the use of KADS to develop large application systems are virtually nonexistent. We used KADS to model the task of cancer-chemotherapy administration performed by ONCOCIN, a large medical expert system developed at Stanford University in the 1980s. Based on a knowledge-level description of ONCOCIN and on a review of the features of KADS, we developed K-ONCOCIN, a model of the ONCOCIN cancer-chemotherapy task that can be understood in terms of the four layers required by the KADS approach.

In this paper, we offer a detailed description of the elements and layers of the K-ONCOCIN conceptual model. We also describe briefly a design model that implements K-ONCOCIN as a functional expert system. The K-ONCOCIN model provides a data point for understanding the strengths and limitations of the KADS method when applied to a complex domain task.

1. INTRODUCTION

Workers in artificial intelligence (AI) have used a variety of models to analyze successful expert-system applications (Clancey, 1985) and to provide the foundations for future knowledge engineering (McDermott, 1988). Combined with algorithms and data structures, these models can provide structuring mechanisms for the representation and use of knowledge (Chandrasekaran, 1988). Models and their application to knowledge engineering have become topics of research (Musen 1989a; Karbach, Linster & Voß, 1990). The diversity of the modeling approaches that are discussed in the literature (e.g., Clancey, 1985; Alexander et al., 1987; Breuker et al., 1987; Shaw & Gaines, 1987; Chandrasekaran, 1988; Marcus, 1988; McDermott, 1988; Musen, 1989c; Boose, Shema & Bradshaw, 1989; Gruber, 1989; Punch & Chandrasekaran, 1990; Gaines & Shaw, 1990; Bredeweg et al., 1989) calls for a systematic comparison using common exemplars that can demonstrate how the different approaches relate to one another. In this paper, we use the KADS (Knowledge Acquisition and Design Structuring) methodology developed at the University of Amsterdam (Wielinga & Breuker, 1986; Breuker et al., 1987; Schreiber et al., 1988; Breuker & Wielinga, 1989) to model the task of cancer-chemotherapy administration—the same application task modeled by the ONCOCIN expert system (Shortliffe et al., 1981; Tu et al., 1989).

The KADS approach comprises a promising set of modeling techniques for developing commercial expert-system applications (for example, Schachter-Radig & Wermser, 1988) that also provides the scientific foundation for much current research (Voß et al., 1990). Previous publications on KADS either have focused on limited application problems to illustrate the methodology (Wielinga et al., 1989) or have discussed the approach on an abstract level (Breuker et al., 1987). To provide a sound data point for an evaluation of KADS, we selected ONCOCIN as a reference task. ONCOCIN is a real-life application that has been the basis for considerable work in the area of knowledge acquisition and that has been broadly described and discussed in the literature.

We first give a knowledge-level overview of ONCOCIN. We then provide a brief summary of those aspects of the KADS methodology that are relevant for this paper. Using the terminology of KADS, we then describe K-ONCOCIN—a KADS model of the ONCOCIN task—which forms the basis for our evaluation and discussion of the KADS approach.

2. A KNOWLEDGE-LEVEL ANALYSIS OF ONCOCIN

We first provide an overview of the knowledge-level (Newell, 1982) components and behaviors of ONCOCIN. We analyze ONCOCIN from the perspective of OPAL (Musen et al., 1987), a knowledge-acquisition tool that facilitates development of the ONCOCIN knowledge base. We emphasize the goals, actions, and knowledge

structures of the ONCOCIN system. A technical description of the ONCOCIN implementation has been published elsewhere (Tu et al., 1989).

2.1. Overview of ONCOCIN

ONCOCIN provides an automated mechanism to monitor the treatment of cancer patients assigned to standard treatment protocols. It includes a database of specifications for numerous protocols, and a database of test and treatment records for numerous patients. The system assists the treating physician by providing recommendations according to protocol guidelines for patient therapy and for the requisition of tests . On the basis of (1) the latest laboratory-test results for a particular patient, (2) information from the most recent clinical examination, (3) the patient's treatment history, and (4) the specific treatment protocol to which the patient is assigned, ONCOCIN computes for that patient a recommendation for the next cycle of treatments and tests .
We shall now give a detailed account of the protocol knowledge that ONCOCIN uses to generate its recommendations. This knowledge bears directly on the design decisions for the KADS model of ONCOCIN. Because K-ONCOCIN models a complex application task, we must present a considerable amount of domain-specific knowledge to define the salient features of our model.

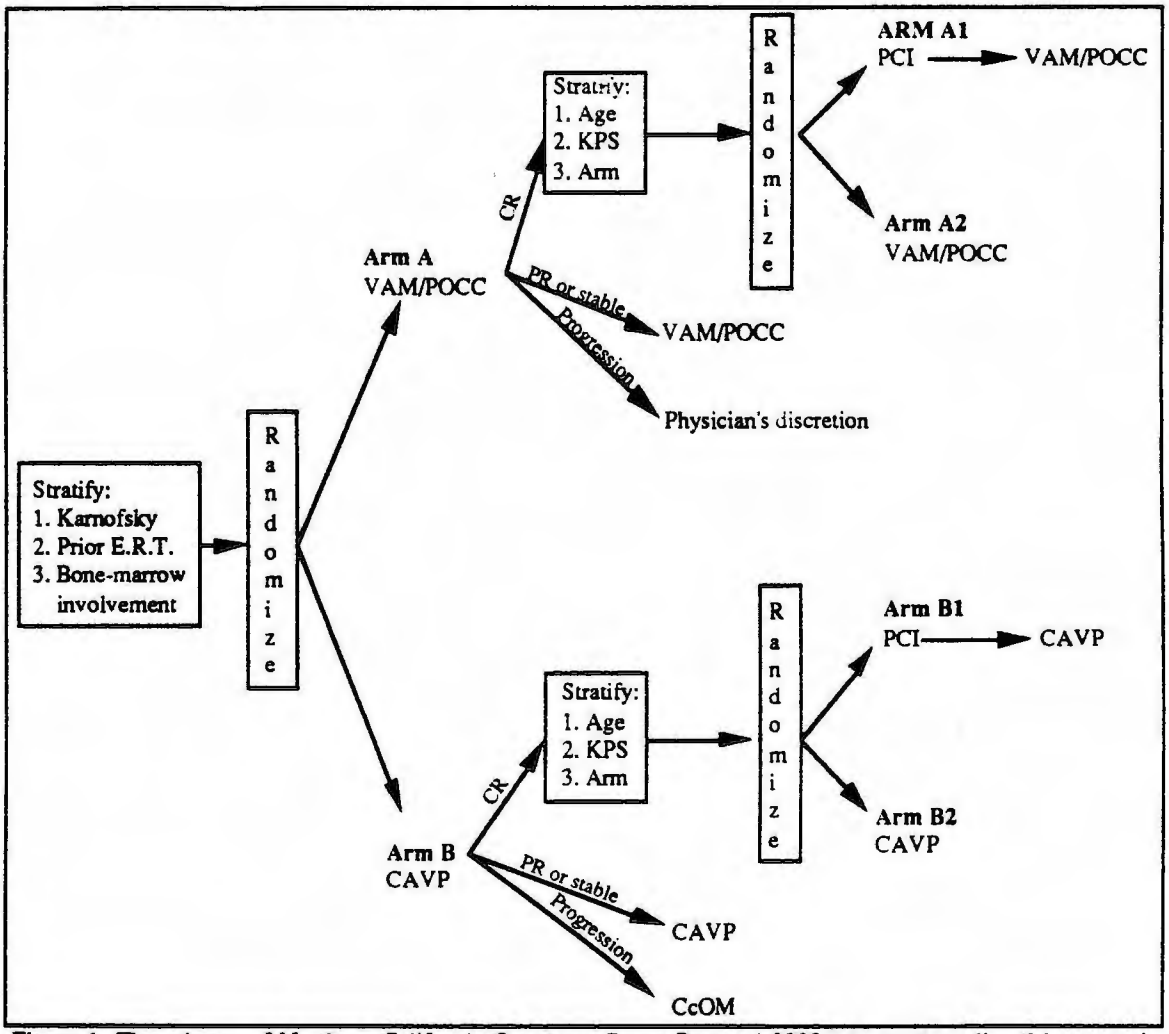


Figure 1: The schema of Northern California Oncology Group Protocol 2083-1 to treat small-cell lung carcinoma. Patients must meet certain preconditions to be enrolled in the protocol. They are stratified according to their Karnofsky status, prior emergency radiation therapy (ERT) and bone-marrow involvement. They are then randomly assigned to arms A or B of the protocol. Arm A analyzes the combination of the chemotherapies VAM and POCC, and the effect of prophylactic cranial irradiation (PCI). Arm B tests the chemotherapies CAVP and CcOM, and the effect of PCI. In both arms of the protocol, patients showing complete response (CR) are stratified again, according to the previous arm of the protocol, their Karnofsky performance status (KPS), and their age, before being randomized into groups that are treated with combinations of radiotherapies and chemotherapies, or just chemotherapies. The treatments vary, according to whether patients demonstrate partial response (PR) or progression of cancer.

2.2. The Specification of a Protocol

A treatment protocol is defined by two kinds of information:

1. Each protocol has a time- and decision-oriented control structure that determines all permissible sequences of tests and treatment cycles (Section 2.2.1).
2. Each test and treatment sequence referred to by the control structure is specified in detail (Section 2.2.2).

2.2.1. The Control Structure of a Protocol

For an individual patient, a protocol treatment consists of a sequence of tests, X-ray therapies, and chemotherapies. The control structure of the protocol determines all permissible sequences of tests and treatments for any patient enrolled in the protocol (Figure 1). A variety of sequences is provided by the incorporation into the control structure of branching logic that depends on the results of particular tests or clinical assessments, and by branch points that represent situations in which patients are assigned at random to alternative forms of treatment as part of the clinical experiment embodied in the protocol.

The diagram in Figure 1 describes the general schema of Northern California Oncology Group Protocol 2083-1 in terms of chemotherapies, X-ray therapies, tests, and randomizations (Carlson et al., 1986).

2.2.2. Specification of Tests and Treatments

Each protocol includes a list of the initial laboratory tests and the tests that are to be repeated during and after the treatment sequence. The information about each repeated laboratory test includes the frequency of testing and various disease- and treatment-related conditions that indicate (or contraindicate) the performance of the test.

An X-ray therapy is specified by the total radiation dosage, the number of patient visits, the dosage at each visit, and the period of time over which the therapy is to be administered. If the X-ray therapy is to interrupt a chemotherapy, the point of interruption is specified.

A chemotherapy is specified as (1) a number of cycles and subcycles of various drug treatments, and (2) the laboratory tests to be administered on specific days of each cycle. (A cycle is simply a particular course of drug treatment administered for a particular number of days.) In cases of severe drug toxicity, of abnormal laboratory-test results, or of abnormal clinical symptoms, the specification includes details of treatments to be administered during delays to (or on termination of) the standard treatment cycles. Also specified are both the time that must be allowed between the end of any prior X-ray therapy and the start of the therapy, and the maximum number of consecutive treatment delays that may occur during the therapy.

The specification of a drug treatment indicates, for each drug in the treatment, the normal dose (expressed either in absolute terms or in relation to some patient characteristic, such as body weight), the route of administration, the duration of the administration of one dose, the number of administrations, the frequency of administrations, the days in the cycle on which administrations are to be made, the maximum individual dose, and the maximum cumulative dose. In addition, the specification indicates actions to be taken if the degree of drug toxicity is too high, if the patient's white-blood-cell count or platelet count falls within certain ranges, or if other specific laboratory-test results are abnormal. The actions applicable in such circumstances may include an attenuation of the dosage of some drug(s), a substitution of another specified drug treatment, a specified break in treatment, a termination of the current cycle (in which case a special drug treatment may be used before the start of the next cycle), a skip to the next cycle (for alternating chemotherapies), a switch to another named protocol, the removal of the patient from any treatment protocol, reporting of the result to the person who developed the protocol, and a consultation with other physicians. Protocols also may specify dosage-attenuation rules for the use of drugs in cycles that *follow* delayed and aborted cycles, and for the administration of drugs subsequent to X-ray therapy.

A rule for attenuating the dosage of a drug may express the recommended dose as a percentage of either the normal dose or the previous (possibly already attenuated) dose. Whenever more than one attenuation rule may be applicable, there are rules to specify whether the recommended attenuation should be based on the combination of factors (multiplication of the attenuation fractions) or whether a single attenuation should be applied. For each drug in each treatment, there may be specified a minimum attenuation percentage that represents the level at which it is more appropriate to suspend the cycle than to continue reducing the dosage. The indication for such a delay will be overridden if the maximum number of delays for the therapy has already been reached. In that case, a minimum default attenuation is applied. A special rule is used to round off the calculated dose to an appropriate unit for each specific drug (e.g., tablet or vial size).

2.3. The Patient Database

Information stored for each patient includes the name of the assigned protocol and a time-stamped record of each clinic visit. A visit record contains the results of laboratory tests and of clinical examinations conducted since the previous visit, and an account of X-ray or drug treatments administered at the recorded visit.

3. KADS

The KADS methodology initially was developed in ESPRIT Project P1098 to provide methodological support for the industrial development of knowledge-based systems. We review the methodology briefly. As KADS is still under development, and as many aspects have not been spelled out in great detail, the following section is certainly biased by our backgrounds and by our personal interpretations of aspects of the KADS approach.

3.1. The Methodology

KADS is a methodology for the structured and systematic development of knowledge-based systems, which aims to provide software-engineering support for the knowledge-engineering process—from the first exploratory interviews of experts to the development of an operational system. KADS is a manual, non formal method. Work is underway to provide a KADS workbench of computer-based tools that will assist knowledge engineers in applying the KADS principles (Anjewierden & Wielemaker, 1990).

KADS divides the development process of a knowledge-based system into a suite of models that represents the transformation of knowledge from the initial, informal phases into an operational system. Within the models, knowledge is represented on several layers to stress the different types of expertise that developers must represent within a knowledge-based system.

Table 1: The four layers used within KADS to describe different types of knowledge (Source: adapted from Breuker & Wielinga 1989, p. 277).

Level	Relations	Objects	Organization
Domain		Concepts, relations structures	Axiomatic structure
	Describes		
Inference		Meta-classes, knowledge sources	Inference structure
	Applies		
Task		Goals, tasks	Task structure
	Controls		
Strategy		Plans, meta-rules, repairs, impasses	Process structure

Table 2: A typology of knowledge sources, with the kind of operations they perform, and their input and output arguments (Source: adapted from Wielinga, Schreiber & Breuker, 1991, p. 22).

Operation type	Knowledge source	Arguments
Generate concept/instance	Instantiate	Concept → instance
	Classify	Instance → concept
	Abstract	Set of instances → concept
	Specify	Concept → concept
	Select	Set → concept
Change concept	Assign-value	Attribute → attribute-value
	Compute	Structure → attribute-value
Differentiating values/structures	Compare	Value + value → value
	Match	Structure + structure → structure
Structure manipulation	Assemble	Set of instances → structure
	Decompose	Structure → set of instances
	Transform	Structure → structure

3.2. The Layers of Knowledge in KADS

In all KADS models, four types of knowledge may be distinguished: domain-layer, inference-layer, task-layer and strategy-layer knowledge. Table 1 shows the four layers, their vocabulary, and their interaction. The *domain layer* describes all the knowledge of the application domain in the form of concepts, structures, and relations. The representation of the domain-layer knowledge is independent of the inferences that are to be made with the domain-layer knowledge by the problem solver. The *inference layer* describes the problem-solving capabilities of a system in terms of *knowledge sources* and *meta-classes*. Meta-classes describe roles and states in the problem-solving process. Knowledge sources are functional descriptions of inferences that operate on meta-

classes. The combination of meta-classes and knowledge sources forms a domain-independent data-dependency diagram: the *inference structure*. Table 2 shows a typology of knowledge sources proposed by the developers of KADS in (Wielinga, Schreiber & Breuker, 1991, p. 22).

3.3. The Suite of Models in KADS

The KADS methodology provides a suite of models that leads from raw data (results of interviews, case descriptions) to operational code (Figure 2). The four layers described in Section 3.2 can be found in each of the models. From data gathered from interviews, protocol analysis, and background literature, the knowledge engineer first builds an informal model: the conceptual model (M1 in Figure 2). This model accounts for the observed expertise using the four layers of knowledge. To facilitate the interpretation of the data, KADS provides a library of interpretation models that consist of a task and an inference layer. These interpretation models serve as templates in the development of the conceptual model. A KADS interpretation model provides a domain-independent framework to help the knowledge engineer with the analysis, interpretation, and model formation in a new situation that she does not yet understand fully. A hierarchically organized library of interpretation models facilitates the selection of a model. The interpretation models of the library can be adapted to the problem at hand. If no model can be selected, the knowledge engineer builds one from scratch. Experience shows that such model creation is done frequently, especially for application tasks that must be solved using constructive problem-solving methods. An interpretation model consists of the three upper layers of KADS (i.e., the inference, task, and possibly the strategy layers). An interpretation model is a domain-independent, abstract description of a problem-solving method. It gives the knowledge engineer strong indications of what domain knowledge is needed, because the meta-classes and knowledge sources indicated in the interpretation model must have domain-specific equivalents.The conceptual model is transformed into a design model (Bredeweg et al., 1989) that specifies the high-level programming techniques that will be used to implement the elements of the conceptual model (M2). At the same time, external requirements such as modularity and speed influence the design model (M3). The design model can then be transformed by way of the detailed design model into operational code.

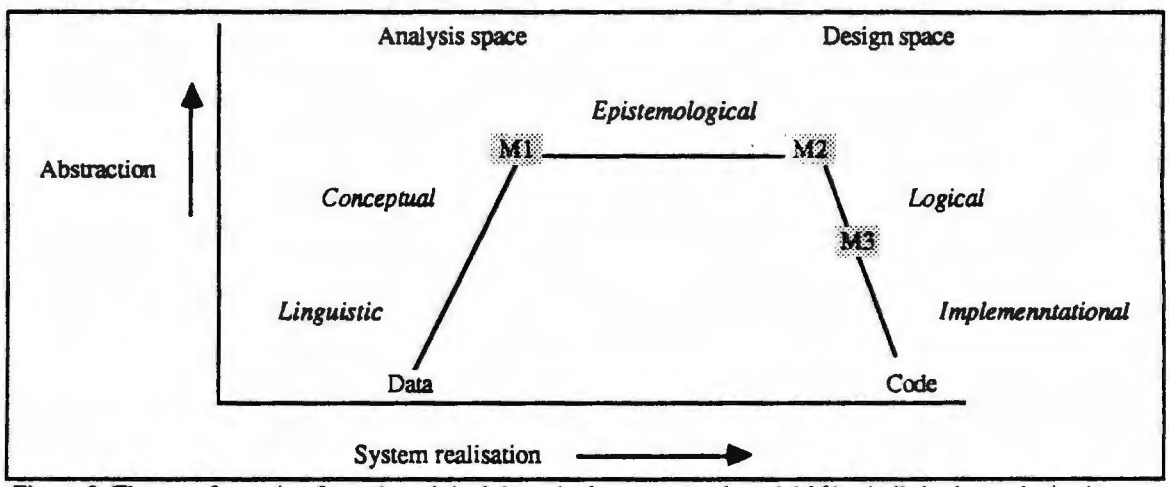


Figure 2: The transformation from the original data via the conceptual model M1—built in the analysis phase—the design model M2, and the detailed design model M3 ultimately results in operational program code. The labels in *italics* are Brachman's (1979) levels of analysis (Source: adapted from Breuker & Wielinga, 1989, p. 268).

4. THE KADS MODELS OF THE ONCOCIN TASK

In this section, we describe the development of K-ONCOCIN—the model of ONCOCIN in KADS' terms. We focus our description on the conceptual model of K-ONCOCIN. We briefly present the design model; a presentation of the implemented system is beyond the scope of this paper.

We use NCOG (Northern California Oncology Group) Protocol 2083-1 (Carlson et al., 1986) to illustrate K-ONCOCIN (Protocol 2083-1 has served the purpose of illustrating aspects of ONCOCIN and OPAL in a number of other publications; Musen et al., 1987; Musen, 1989b).

Figure 1 shows the schema for Protocol 2083-1. This protocol describes a clinical trial that measures the effectiveness of a new, simplified chemotherapy (called CAVP because it includes the administration of the drugs Cytoxan, Adriamycin, vincristine, and predisone) in treating small-cell cancer of the lung. The chemotherapy is compared with the standard treatment for this tumor (two alternating chemotherapies, called POCC and VAM, which require the administration of seven different drugs in total). If a patient fails to respond to CAVP, the protocol specifies that he should receive another chemotherapy, called CCoM. When physicians carry out this medical experiment, the patients who have small-cell lung cancer must give their consent to be assigned at random either to Arm A of the protocol (standard therapy with POCC and VAM) or to Arm B of the protocol (experimental therapy with CAVP and, if necessary, CCoM). A second experiment within Protocol 2083-1 tests

the effectiveness of prophylactic cranial irradiation (PCI) in patients who show complete response (CR) to their assigned drug treatments (see Figure 1). Protocol 2083-1 is just one of 36 protocols currently encoded in the ONCOCIN knowledge base.
Note that we limit K-ONCOCIN to that part of ONCOCIN that is accessed and used by OPAL-generated knowledge bases. Because of this restriction, certain aspects of ONCOCIN are not part of K-ONCOCIN.

4.1. Use of KADS to Develop the Conceptual Model

This section describes how we obtained the conceptual model for K-ONCOCIN. We believe that it is necessary to include a description of the model-building process to illustrate how we used KADS as a support for the model-building activity. A complete description of every step of the conceptualization is outside the scope of this paper. We thus limit our presentation to the modeling of the iteration through the therapies during a treatment—a part of the modeling process that has been documented completely on electronic media. This record allows us to reconstruct the process with sufficient detail to show the interaction of the KADS layers during modeling.
Most of the knowledge that we used for the development of the KADS model was contained in documents: the description of the NCOG Protocol 2083-1 (Carlson et al., 1986), a functional specification of OPAL (Combs, 1988), and the printouts of the OPAL data structures for Protocol 2083-1. The printouts consisted of listings of the frame hierarchy and of the transition network that OPAL provides for the ONCOCIN Reasoner. These documents relate to one protocol. This restriction means that our modeling process may have been biased, as we did not analyze a representative sample of all the protocols that ONCOCIN handles currently. Furthermore, the frame hierarchy and the transition network reflect the model of protocol-based oncology treatment created by the original knowledge engineers. We inferred our model—K-ONCOCIN—from the code reflecting the original developer's conceptualization.

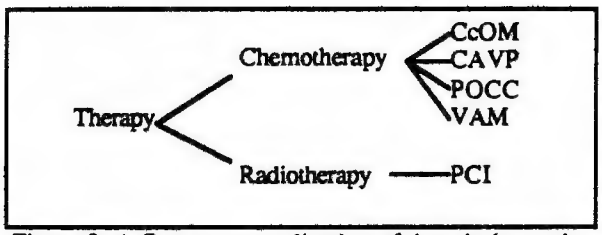


Figure 3: A first conceptualization of domain-layer elements. The hierarchy describes chemotherapies and radiotherapies.

For the development of K-ONCOCIN, we tried to delimit our professional roles. One of us (Musen), who had considerable knowledge about OPAL, ONCOCIN, and chemotherapy administration, served in the role of the domain expert; the other (Linster), who had previous experience with KADS-based development of expert systems, concentrated on the role of the knowledge engineer. We avoided some of the communication problems that are characteristic of knowledge-engineering situations, because Musen, through his professional experience could bridge the gap that normally exists between the conceptual model a domain expert has of his real-world task, and the conceptual model that a knowledge engineer must develop in the process of building a knowledge-based system. Obviously, as we both were interested in the modeling process and methodology, we did not maintain this role order after the initial developments.

4.1.1. The General Procedure

The KADS layers are relatively independent of one another. They represent several different views of a real-world behavior. Each view contributes to a constructive process of making sense of elicited data—a process that eventually results in a coherent model. Therefore, in the development of a KADS conceptual model, the knowledge engineer moves among the layers to exploit the interactions between the different points of view. For example, an initial analysis of the *domain* provides insight into the interrelations and dependencies among domain concepts, which in turn allows the knowledge engineer to perceive recurring *inference* patterns. In new situations, these inference patterns function as anticipatory structures that focus the knowledge engineer's attention. She must look for the patterns' domain equivalents in transcripts of new protocols, in background texts, or in video recordings. Similar interactions exist with the *task* layer. The task layer is a means to decompose a real-world task into appropriate subtasks. A task decomposition allows the model-builder to develop a structured overview of the situation, and of the behaviors to be modeled. This perspective, in turn, helps her to identify individual concepts and relations on the domain layer.

4.1.2. Modelling of the Iteration Through the Therapies

We based our development of K-ONCOCIN on Protocol 2083-1 and on material directly related to that protocol. The NCOG protocol document is difficult to understand for someone outside the medical profession, and there is no direct and obvious way to build a model from it. The medical document does not explicate the knowledge such that the latter can be used directly in a modeling process. On the other hand, the code of the OPAL representation of the protocol consists of decontextualized knowledge units that derive their semantics from the ONCOCIN inference engine. Thus, in the modeling process, we had to use this interaction between formalized data structures representing bits of knowledge out of context, and textual information, designed for a clinical audience with a different purpose.

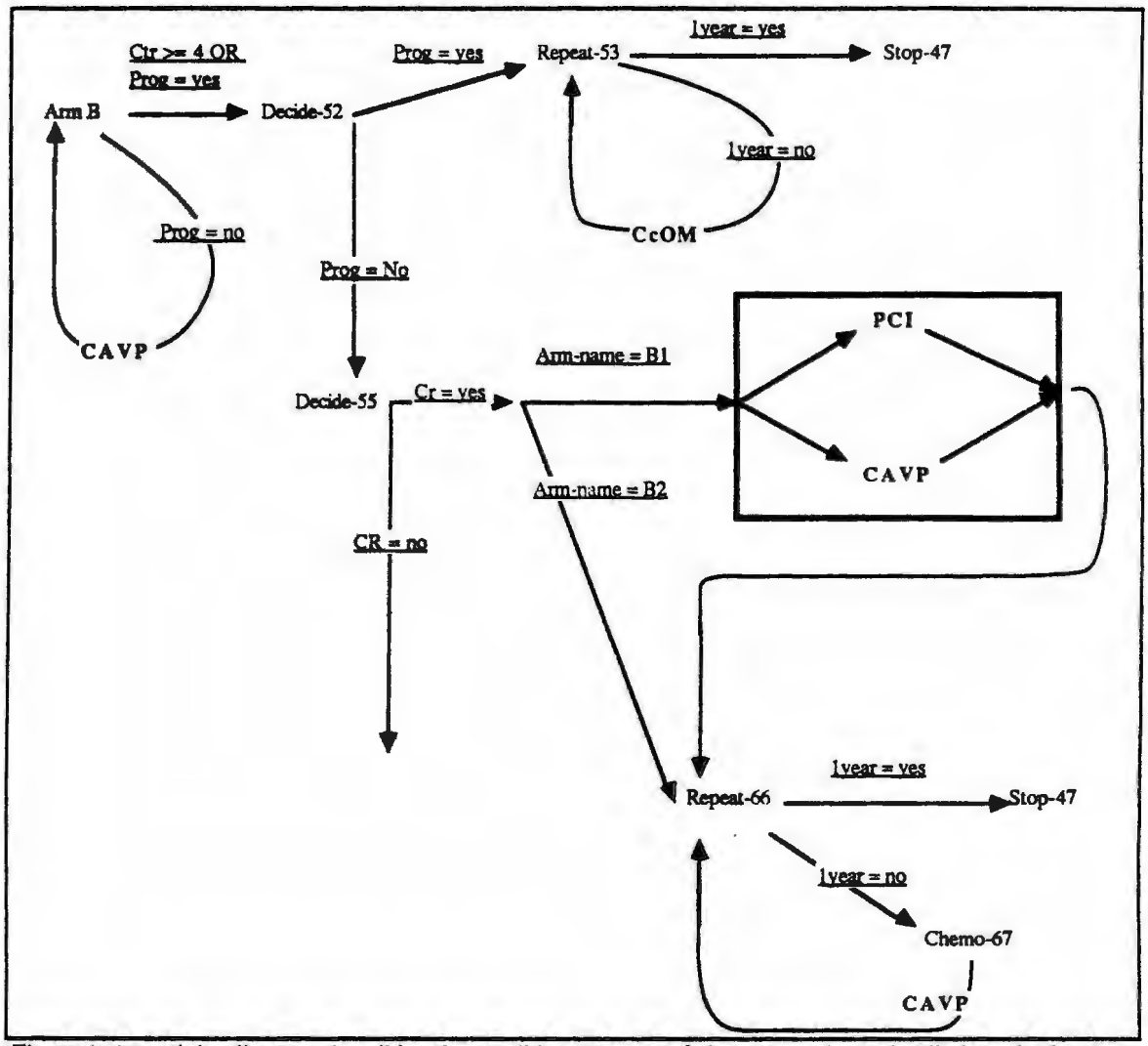


Figure 4: An activity diagram, describing the possible sequences of chemotherapies and radiotherapies in a part of arm A of Protocol 2083-1. Therapeutic actions (e.g., chemotherapies or radiotherapies) are set in **bold**. Branching conditions are underlined. Combinations of therapies that are administered in parallel are boxed. The diagram contains several explicit OPAL-generated intermediary states, such as "Decide-52."

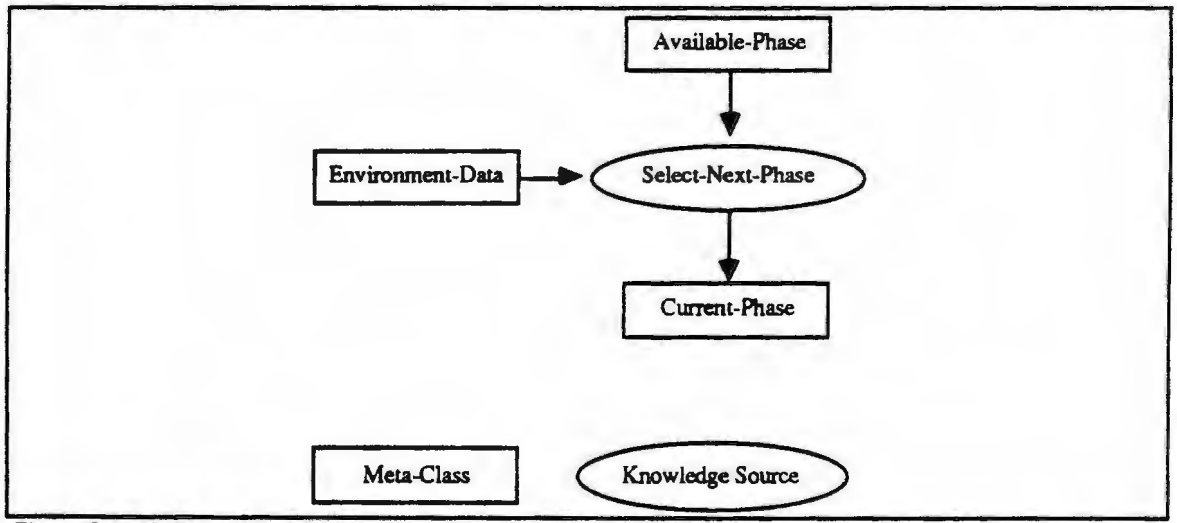


Figure 5: An inference pattern that captures the activities of Figure 4.

In the process of building a conceptual model, we analyzed the domain-layer first, to develop an initial inference structure that would help us in the analysis of the next aspects of the domain. We started out by analyzing the frame hierarchies in OPAL. We used the results to build initial structures of concept definitions (see Figure 3) for the domain layer of K-ONCOCIN. These concept definitions contributed the terminology for the analysis of the transition networks that OPAL provides for ONCOCIN. To clarify the states in the transition networks, we drew activity diagrams (see Figure 4).

On the basis of the activity diagrams, we defined recurring inference patterns to derive more abstract descriptions of the domain-layer phenomena. According to the KADS methodology, these inference patterns must be phrased in domain-independent terms so that they will be reusable. The phrasing of these patterns requires interpretation of raw data. This makes it an iterative process that involves considerable uncertainty. We viewed this pattern-finding activity as a constructive, bottom-up task, not as a top-down, recognition task. According to KADS, the patterns are phrased in terms of knowledge sources and meta-classes. We thus labeled the knowledge sources with verbs or verb phrases, to emphasize their change-inflicting character. The meta-classes were tagged with nouns or noun phrases.

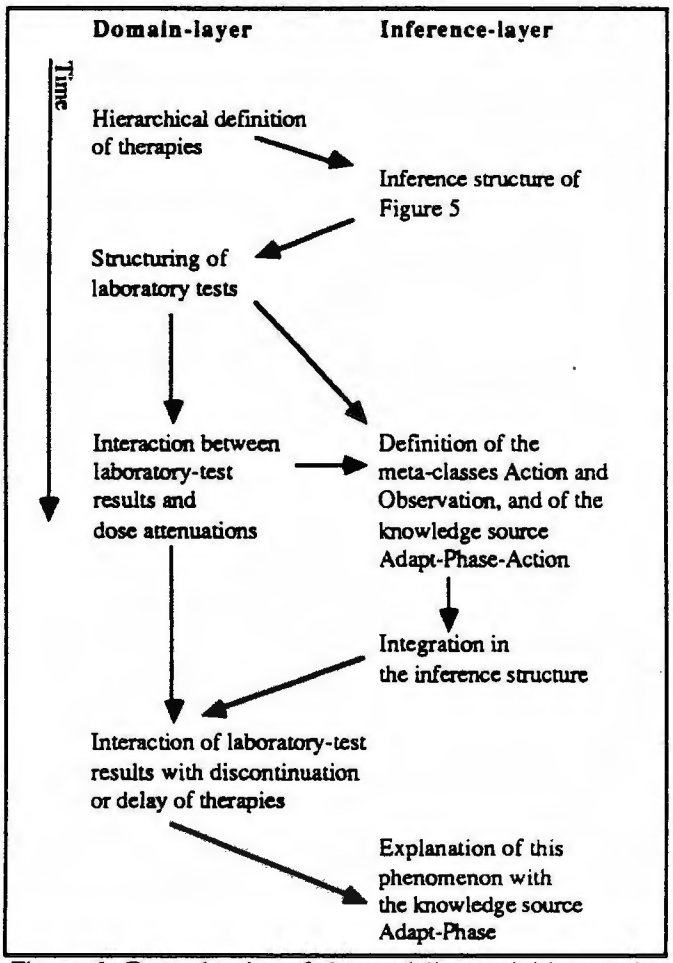


Figure 6: Dependencies of the modeling activities on the domain and inference layers during the development of K-ONCOCIN.

For example, in Figure 4, we can identify several activities that have a temporal extension, such as CAVP, CcOM, and so on. From the inference-layer view, the label *phase* accounts for the activities' temporal facet. Other elements of the activity graph can be interpreted as choice points, where the problem solver determines the subsequent phase. From an inference-layer point of view, they are the domain-layer equivalents of a knowledge source that selects the next active phase. Because all phases are known to the system in advance (i.e., no phases—therapies in our application—are created dynamically), it seemed reasonable to talk about the following two roles that therapies can play in K-ONCOCIN: (1) the meta-class *Available-Phase* denotes all the therapies that are available in the model, and (2) the meta-class *Current-Phase* points out which therapy is being administered currently. Obviously, the decision process selecting the next current phases must take in account which phases are available.

Besides knowledge about the available phases, the decision for the next phase requires more information that makes it sensitive to the environment. Looking at the activity graph of Figure 4, we can see that the decisions to determine the next current phase are taken on the basis of several parameters—besides the available phases—that describe the patient's current state, such as the progression of the cancer (e.g., *progression = yes*, or *complete response (CR) = yes*), the number of times a certain therapy has been administered (e.g., *Ctr > 4*), or the patient's total time on the protocol (e.g., *1year = yes*). We selected the label *Environment-Data* as a suitable, domain-independent name for the meta-class denoting these parameters.

Figure 5 shows the KADS inference structure of the interaction of the meta-classes *Available-Phase* (i.e., all the therapies included in the model), *Current-Phase* (i.e., the therapy that is presently being administered) and *Environment-Data* (i.e., the data about the patient's current state) with the knowledge source *Select-Next-Phase*[1]. This inference structure is an abstract description of the recurrent inference patterns observed in Figure 4. The meta-class *Environment-Data* points to data about the patient, especially to parameters describing the cancer's response to the treatment. This focused our attention on the analysis of the roles that the laboratory-test

[1]In many KADS models that incorporate a knowledge sources of the type *select*, the set from which the knowledge source selects, is not mentioned explicitly. We assume however, that a model should explicate all the input and output relationships of its building blocks.

results play in the inference process. Interesting enough, no laboratory-test result or other direct input of patient data plays a role in the problem-solving process up to now. This obvious lack of input led us to the next step in the model-building process (see Figure 6), where we looked at the available tests in a protocol, and at their influence on medication and therapy. The insertion of the laboratory tests was a next step in defining the remaining domain structure (see Figure 8), and the whole inference structure (see Figure 12).
There was no need to perform a systematic task analysis in our construction of the K-ONCOCIN conceptual model. The developers of the original ONCOCIN and OPAL systems had already identified the subtasks required for cancer-chemotherapy administration (Tu et al., 1989). Because there was no interpretation model for skeletal-plan refinement in the KADS library to help us create a new task decomposition for K-ONCOCIN, it was expedient to adopt the task decomposition that was already modeled in OPAL (and that had been regenerated using PROTEGE; Musen, 1989b). We identified a straightforward mapping from the knowledge sources that we had modeled on the inference layer to the task decomposition that had been built into OPAL.

4.2. The Conceptual Model

This section describes the KADS conceptual model for ONCOCIN. We detail the representation of the domain-layer knowledge, then describe the inference-layer elements and their links to the domain layer. The task-layer knowledge of K-ONCOCIN will be described last.

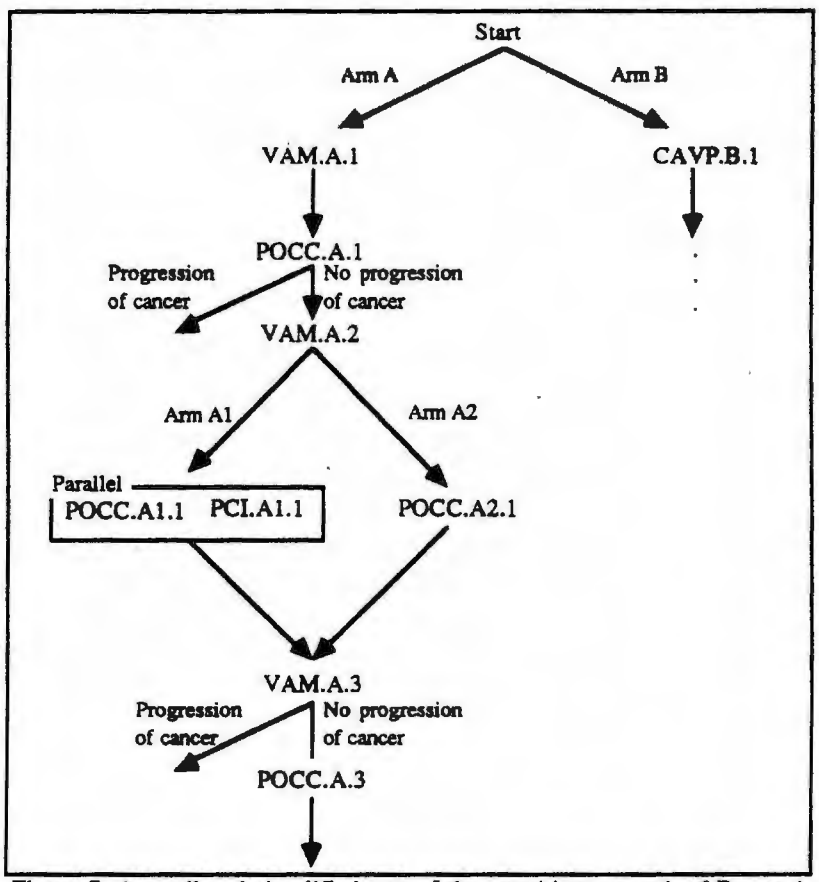


Figure 7: A small and simplified part of the transition network of Protocol 2083-1. The nodes in the network correspond to therapy applications, such as VAM.A.1. "VAM.A.1" stands for the first occurrence of the chemotherapy VAM in Arm A of the protocol. The node Start is a dummy node. The arrows between therapies represent the relation successor.

4.2.1. The Domain Layer

KADS requires a declarative representation of the domain-layer knowledge in terms of concepts, relations, and structures. KADS does not include special representation formalisms, such as transition nets for temporal relations, PROLOG clauses for assertions, or object structures for factual knowledge. Within the general framework of concepts, relations, and structures, the knowledge engineer thus has considerable freedom—and little guidance—when she defines the elements of the domain layer.
We chose an object-oriented approach for the representation of domain knowledge. For the representation of the domain-layer concepts, we use hierarchical definitions of entities with attributes. Relations among entities model knowledge about how the concepts pertain to one another. Our choice of representation formalisms was influenced strongly by our experience with object-oriented knowledge representations as used in PROTÉGÉ, ONCOCIN, and BABYLON (Musen, 1989b; Guesgen, Junker & Voß, 1987; Christaller, Di Primio & Voß, 1989). Users of KADS who have different backgrounds (such as PROLOG or OPS-5 programmers) probably will choose to use different representations.
For the specification of the temporal iteration of the therapies (the high-level algorithm of a treatment protocol shown in Figure 1), we chose to use a transition-network representation. The specification includes a *successor* relation, which is defined on unequivocal pointers to the therapies (i.e., the radiotherapies and the chemotherapies) and on the conditions that allow for a transition between therapies (e.g., random assignment of a patient to a particular arm of the protocol or progression of cancer that requires a change in chemotherapy). Figure 7 illustrates the successor relation.

We represent therapies as concepts that have attributes that define additional knowledge about the therapies, such as the therapy's duration, the drugs administered, the laboratory tests required, the maximum number of delays, and so on. For example, in Protocol 2083-1, the chemotherapy *VAM* lasts 21 days; the drugs *VP-16*, *Adriamycin* and *methotrexate* are administered as parts of *VAM*, and two delays of 7 days each are tolerable if a patient has an unfavorable response to the treatment. Figure 8 shows the hierarchical definition of the concepts in the domain layer of K-ONCOCIN.
For each prescription, we define attributes describing on which day and in which therapy that prescription is used. Similar knowledge is represented for the laboratory tests used to evaluate how the patient responds to the therapy.

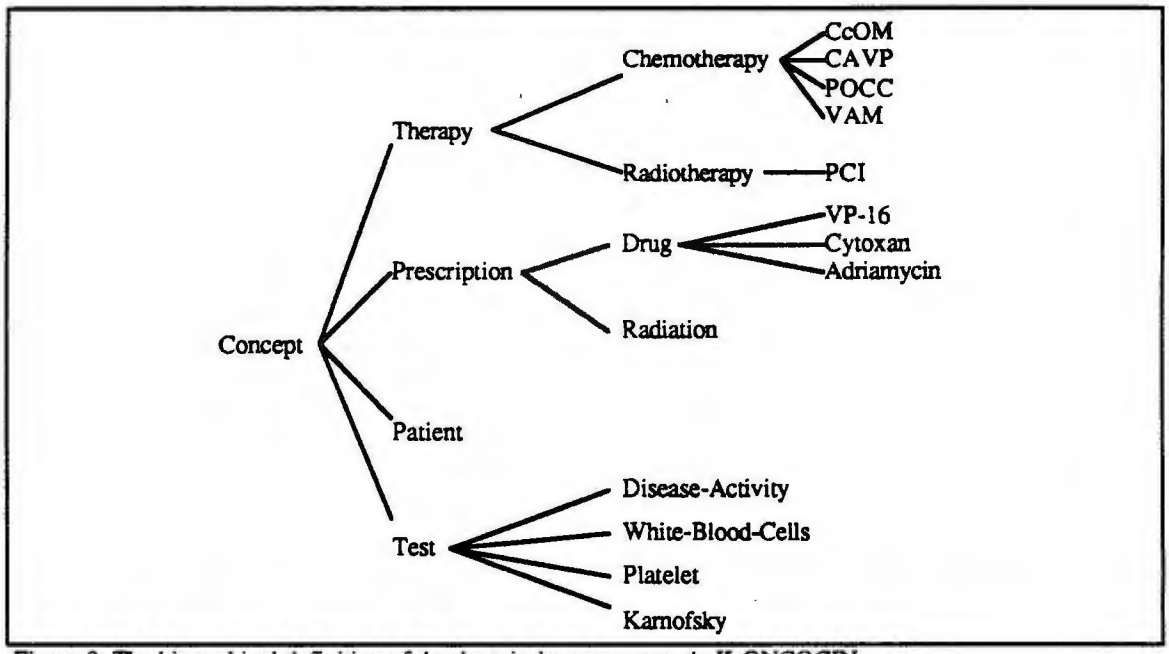


Figure 8: The hierarchical definition of the domain-layer concepts in K-ONCOCIN.

In KADS, knowledge that pertains to several concepts can be represented either as attributes of concepts (i.e., binary relations) or explicitly as relations between concepts. KADS does not give any directives on how to choose an attribute-based or a relation-based representation.
To develop K-ONCOCIN, we represented the following knowledge as relations among concepts:

1. *Drug-in-Chemotherapy* indicates the days of particular chemotherapies on which particular drugs are administrated (e.g., *VP-16* is due on the first day of the chemotherapy *VAM*).
2. *Test-Due* indicates the days of a particular chemotherapy on which a particular test is warranted (e.g., a *WBC* is due on the first day of the chemotherapy *VAM*).
3. *Drug-Dose-Attenuation* relates the results of laboratory tests to the dosage of drugs (e.g., in Protocol 2083-1, the dose of the drug *VP-16* in the chemotherapy *VAM* is reduced to 75 percent of the normal dose if the *WBC* is below $3.5 * 10^3$; if the *WBC* is above $3.5 * 10^3$ and the *platelet count* is between 100 and $150 * 10^3$, then the patient should receive 75 percent of the standard dose of *Cytoxan* in the chemotherapy *CAVP*).
4. *Chemotherapy-Alteration* relates the results of laboratory tests to the continuation or interruption of the current chemotherapies (e.g., if the patient's *platelet count* is less than $100 * 10^3$, then the current chemotherapy is delayed). The duration of the delay is a property of the chemotherapy, just as is the maximum number of delays permitted before the chemotherapy is aborted.
5. *Abstract-Disease-Activity* relates descriptions of how the patient responds to the treatment to more abstract findings (e.g., if the *disease activity* (a numerical representation of the state of the patient's tumor over time) is equal to 6, then the patient is considered not to be responding to treatment, and the cancer is considered to be progressing; if the value of *disease-activity* is equal to 1, then the patient is considered to show complete response to the treatment, and the cancer is considered not to be progressing).

We must emphasize that these entities and relations constitute but one way to view the domain-layer elements of K-ONCOCIN in the KADS framework. Because KADS is not prescriptive in the choice of formalisms for the domain layer, there are many other ways to model the same information.

4.2.2. The Inference Layer

Based on our analysis of ONCOCIN's behavior and on the domain-layer knowledge described earlier, we shall define the elements of the inference layer for the conceptual model of K-ONCOCIN—the knowledge sources, the

meta-classes, and the inference structure. The inference-layer elements are defined in domain-independent terms to make them generic and thus applicable to other domains for which similar problem-solving approaches are appropriate. KADS provides only a vocabulary for the description of general inference-layer knowledge; KADS does not give strong guidance for determining precise names for the individual inference-layer elements themselves.

Many inference structures are known for analytic tasks, such as diagnosis, classification, monitoring, and prediction. ONCOCIN, with its skeletal-plan–refinement problem-solving method, requires a new inference structure, which ultimately should be added to the library of known inference structures for KADS.

In Sections 4.2.2.1 through 4.2.2.3, we define the knowledge sources and meta-classes of K-ONCOCIN. Before explaining how we combined these elements into an inference structure, we show how the inference-layer components are connected to the domain layer.

4.2.2.1. The Knowledge Sources

We discern the following seven kinds of inferences in our model of ONCOCIN's behavior:

1. *Select-Next-Phase* accounts for the inference that selects the next phase in the protocol transition network, based on patient data and on the current phase.
2. *Select-Current-Observation* selects those laboratory tests that should be performed on the current day of treatment.
3. *Abstract-Environment-Data* performs additional inferences to deduce more information from the incoming laboratory-test data, such as those described in the relation *Abstract-Disease-Activity*.
4. *Select-Current-Action* selects the appropriate drugs to administer on the current visit.
5. *Specify-Action-Adaptation* accounts for the modification of the drug dosage based on the results of the latest laboratory-test results and on other clinical data.
6. *Specify-Phase-Adaptation* modifies the current cycle of chemotherapy according to the latest results of laboratory tests. This knowledge source decides whether, if the patient shows signs of drug toxicity, the chemotherapy should be delayed, canceled, or replaced by another drug combination .
7. *Select-Noncurrent-Phase* determines whether the current medical action should be discontinued. For example, VAM will be discontinued if it has been active for 21 days.

Table 3: The correspondence between inference-layer knowledge sources and domain-layer knowledge.

Knowledge Sources	Underlying Domain-Layer Knowledge for Protocol 2083-1
Select-Next-Phase	Successor (a relation) Example: The therapy VAM is selected after POCC's first occurrence in arm A of the protocol.
Select-Noncurrent-Phase	Maximum-Nbr-Delays, Days-in-Therapy (attributes of the concept Therapy) Example: VAM lasts 21 days and can have a maximum of 2 delays.
Select-Current-Action	Drug-in-Chemotherapy (a relation) Example: VP-16 is administered on the first day of VAM
Specify-Action-Adaptation	Drug-Dose-Attenuation (a relation) Example: The dose of VP-16 is reduced to 75% if the WBC $< 3.5 * 10^3$.
Abstract-Environment-Data	Abstract-Disease-Activity (a relation) Example: If the disease activity = 6, then the patient is considered not to have responded to the therapy.
Specify-Phase-Adaptation	Chemotherapy-Alteration (a relation) Example: If the platelet count is less then $100 * 10^3$, then the current therapy should be delayed.
Select-Current-Observation	Test-Due (a relation) Example: WBC should be taken on the first day of the chemotherapy VAM.

4.2.2.2. The Meta-Classes

We observe several roles that domain-layer knowledge elements can play in ONCOCIN's reasoning. In K-ONCOCIN, these knowledge roles correspond to eight meta-classes:

1. *Current-Phase* specifies the therapies that are administered currently.
2. *Available-Phase* describes all the chemotherapies that may be relevant in a K-ONCOCIN treatment protocol.

3. *Current-Observation* specifies the laboratory tests that must be performed on the current day of the current therapy (described by the meta-class *Current-Phase*).
4. *Available-Observation* is the descriptor for all the laboratory tests that are known to K-ONCOCIN.
5. *Environment-Data* identifies all the data that relate to the patient under treatment protocol.
6. *Available-Action* denotes the drugs and radiation prescriptions of the K-ONCOCIN domain-layer knowledge.
7. *Current-Action* represents those prescriptions that have been selected for the current day of the current therapy (described by the meta-class *Current-Phase*).
8. *Updated-Action* represents the prescriptions for the current day when they have been adapted to the knowledge about the patients current state.

4.2.2.3. Linkage of the Inference-Layer Elements and the Domain-Layer Knowledge

Tables 3 and 4 show the domain-layer elements used by each knowledge source, and the domain-layer concepts that can be described by specific meta-classes at certain points in the problem-solving process.

Table 4: The correspondence between the inference-layer meta-classes and the domain-layer knowledge.

Meta-Classes	Underlying Domain-Layer Knowledge for Protocol 2083-1
Environment-Data	Patient (a concept)
Available-Phase	Therapy (a concept) Example: VAM, PCI, POCC
Current-Phase	Therapy (a concept) Example: VAM, PCI, POCC
Available-Action	Prescription (a concept) Example: VP-16, Adriamycin, methotrexate
Current-Action	Prescription (a concept) Example: VP-16, Adriamycin, methotrexate
Updated-Action	Prescription (a concept) Example: VP-16, Adriamycin, methotrexate
Available-Observation	Test (a concept) Example: Karnofsky status, WBC
Current-Observation	Test (a concept) Example: Karnofsky status, WBC
Obsolete-Phase	Therapy (a concept) Example: VAM, PCI, POCC

4.2.2.4. The Inference Structure

The combined meta-classes and knowledge sources form the inference structure. The inference structure can be viewed as an abstract data-dependency diagram. Knowledge sources have meta-classes as input and as output. The inference structure of K-ONCOCIN is complex, so we shall present two partial structures before describing all the data-dependency relationships. The first partial structure is concerned with the selection of the appropriate laboratory tests to be performed on a certain day of a selected therapy. The second structure describes the selection of the proper drugs and the adaptation of their dosage.

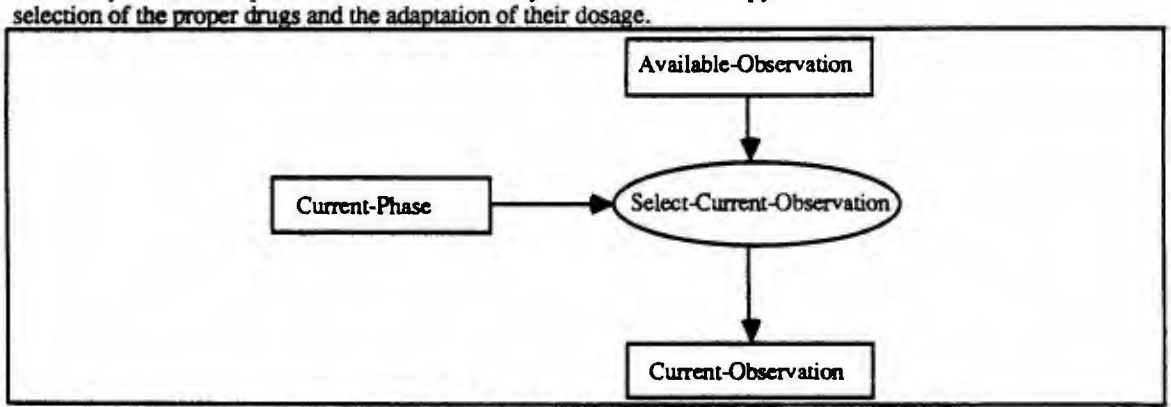


Figure 9: A part of the inference structure of K-ONCOCIN that describes the data dependency among the meta-classes *Available-Observation*, *Current-Phase*, *Current-Observation*, and the knowledge source *Select-Current-Observation*. The knowledge source uses the first two meta-classes to modify the third meta-class.

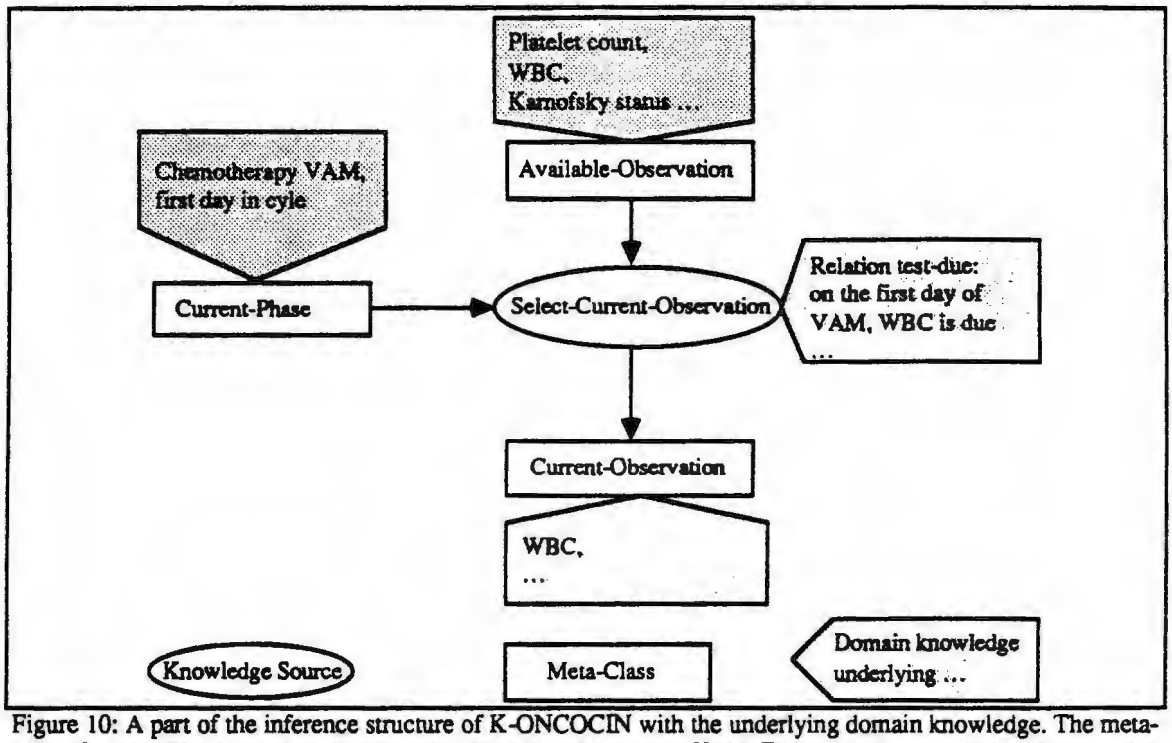


Figure 10: A part of the inference structure of K-ONCOCIN with the underlying domain knowledge. The meta-class *Current-Phase* can describe the role of the domain concept *VAM*. The meta-class *Available-Observation* models the role of the concepts *Platelet count*, *WBC*, and *Karnofsky status*. The knowledge source *Select-Current-Observation* uses the knowledge of laboratory-test ordering described in the domain relation *Test-Due* to modify the meta-class *Current-Observation*.

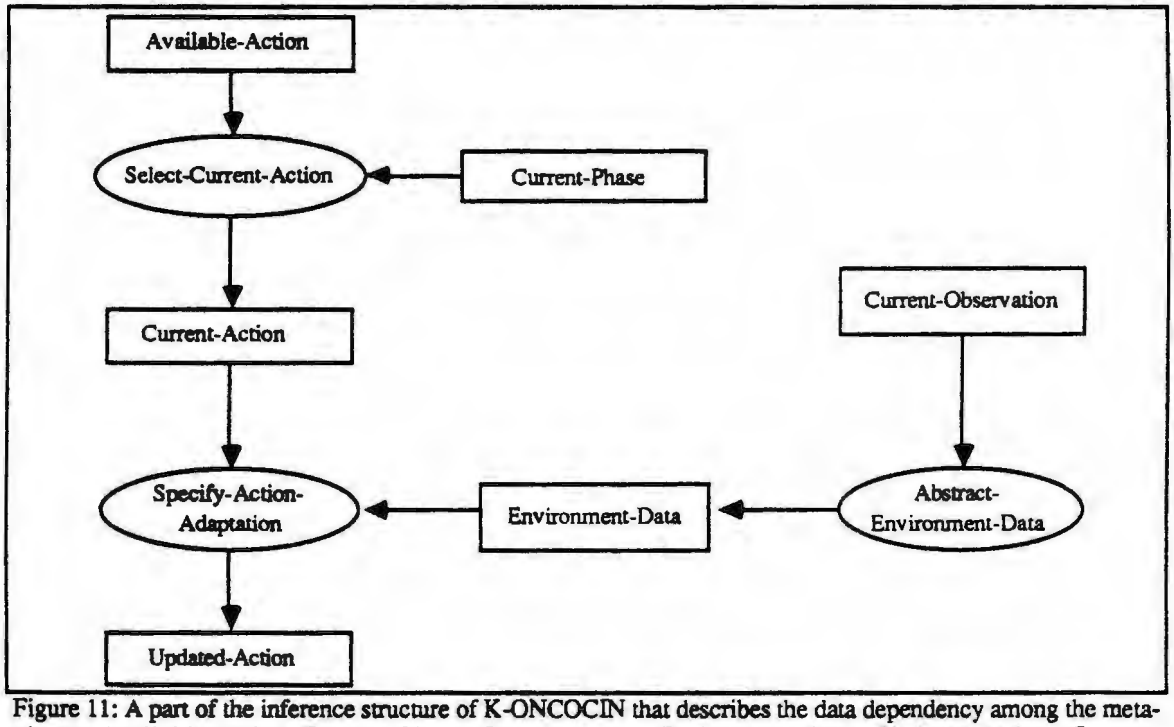


Figure 11: A part of the inference structure of K-ONCOCIN that describes the data dependency among the meta-classes *Available-Action*, *Current-Phase*, *Current-Action*, *Environment-Data*, *Updated-Action*, *Current-Observation* and the knowledge sources *Select-Current-Action*, *Specify-Action-Adaptation*, and *Abstract-Environment-Data*. *Select-Current-Action* works on the meta-classes *Available-Action* and *Current-Phase*; it modifies the meta-class *Current-Action*. *Specify-Action-Adaptation* modifies *Updated-Action* based on *Current-Action* and *Environment-Data*. *Environment-Data* is modified by the knowledge source *Abstract-Environment-Data*, on the basis of *Current-Observation*.

The knowledge source *Select-Current-Observation*, which in K-ONCOCIN identifies the laboratory tests that must be performed based on the current therapy, uses knowledge about the drugs and radiation prescriptions that are available, and uses knowledge about the currently active chemotherapy or radiotherapy. The meta-classes *Available-Observation* and *Current-Phase* describe the required knowledge. The meta-class *Current-Observation* describes the output of the knowledge source. Figure 9 gives a graphical view of this data dependency.
Inference-structure elements are generic descriptions of the problem-solving roles and functions of domain-layer knowledge. Figure 10 shows some of the domain-layer elements that, at a certain point in the reasoning process, may be referenced by the inference-layer elements in Figure 9.

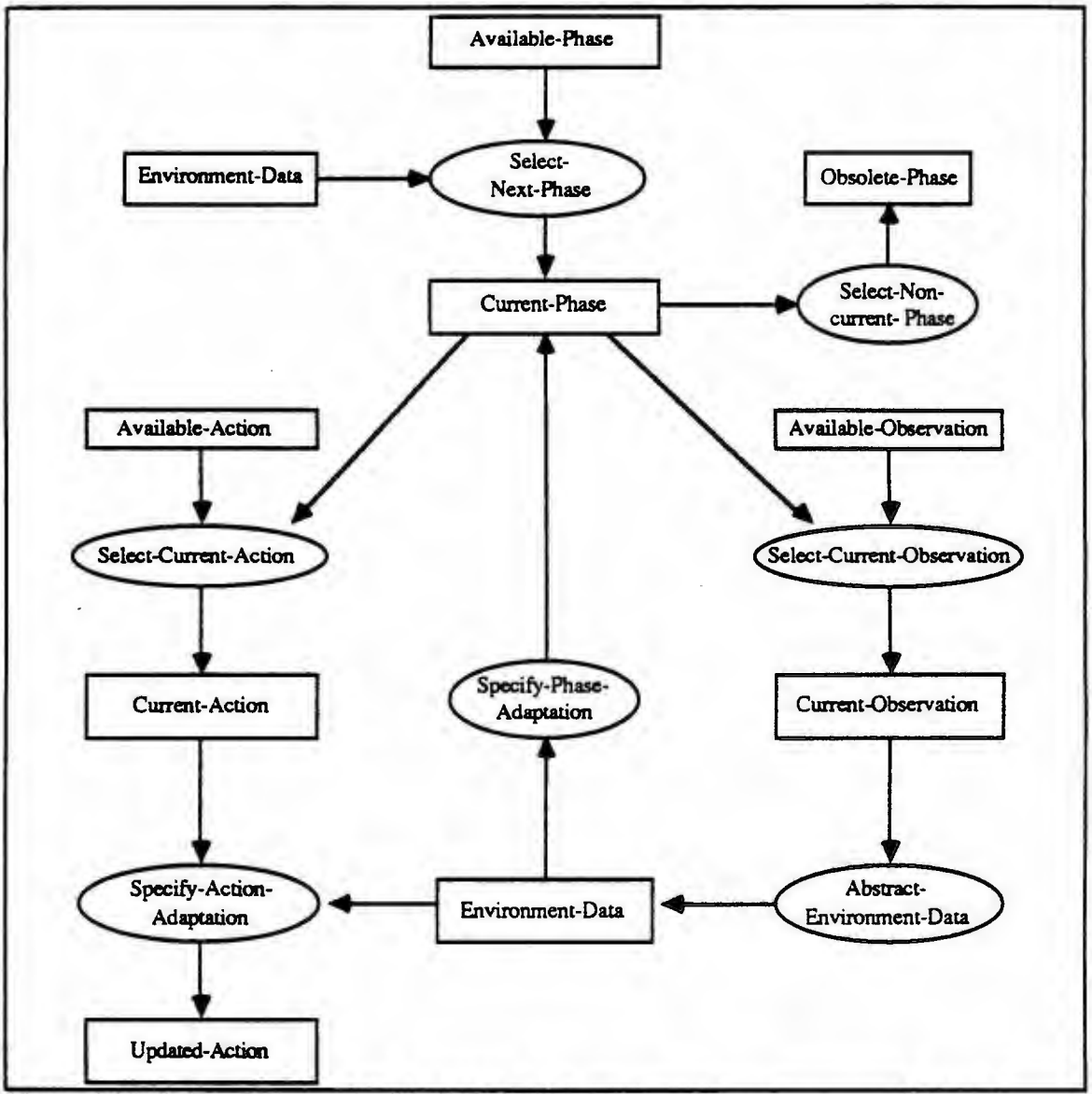


Figure 12: The inference structure for K-ONCOCIN. It combines the inference structures of Figures 9, and 11 with the inference structure described in Figure 5.

In Figure 11, the knowledge source *Select-Current-Action* decides which drugs or radiation treatments (denoted by the meta-class *Available-Action*) should be applied on the current day of the current therapy (described by the meta-class *Current-Phase*). The knowledge source *Specify-Action-Adaptation* takes the selected drugs (which then correspond to the meta-class *Current-Action*) and adjusts the drugs' doses according to the patient's response to previous treatment (as reflected in the meta-class *Environment-Data*). The attenuated drugs are described by the meta-class *Updated-Action*. The knowledge source *Abstract-Environment-Data* uses the patient's test results (denoted by the meta-class *Current-Observation*) to derive additional abstractions that describe the patient's condition, described by the meta-class *Environment-Data*.
By combining knowledge sources with their input and output meta-classes, we finally obtain the complete inference structure of K-ONCOCIN, illustrated in Figure 12. This inference structure combines the partial inference structures of Figures 5, 9, and 11. The knowledge source *Determine-Next-Phase*, described in

Section 4.1.2, controls the use of the transition network that describes the top-level structure of the treatment protocol (see Figure 5).

4.2.3. The Task Layer

The task layer of the K-ONCOCIN model describes how the problem solver must organize its behavior as a sequence of calls to the knowledge sources. Figure 13 shows the top-level goal structure. The subgoal *Evaluate phase* is decomposed further as shown in Figure 14. The leaves of the goal-decomposition tree represent the knowledge sources at the inference layer that satisfy the corresponding subgoals.

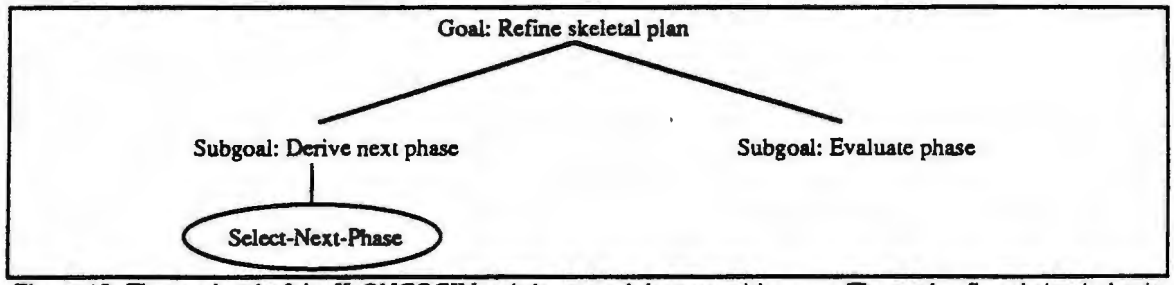


Figure 13: The top-level of the K-ONCOCIN task-layer goal decomposition tree. The goal *refine skeletal plan* is decomposed into the subgoals *derive next phase* and *evaluate phase*. The first subgoal can be fulfilled immediately by the inference-layer knowledge source *Select-Next-Phase*. Satisfying the goal *evaluate phase* requires further subgoal decomposition, as shown in Figure 14.

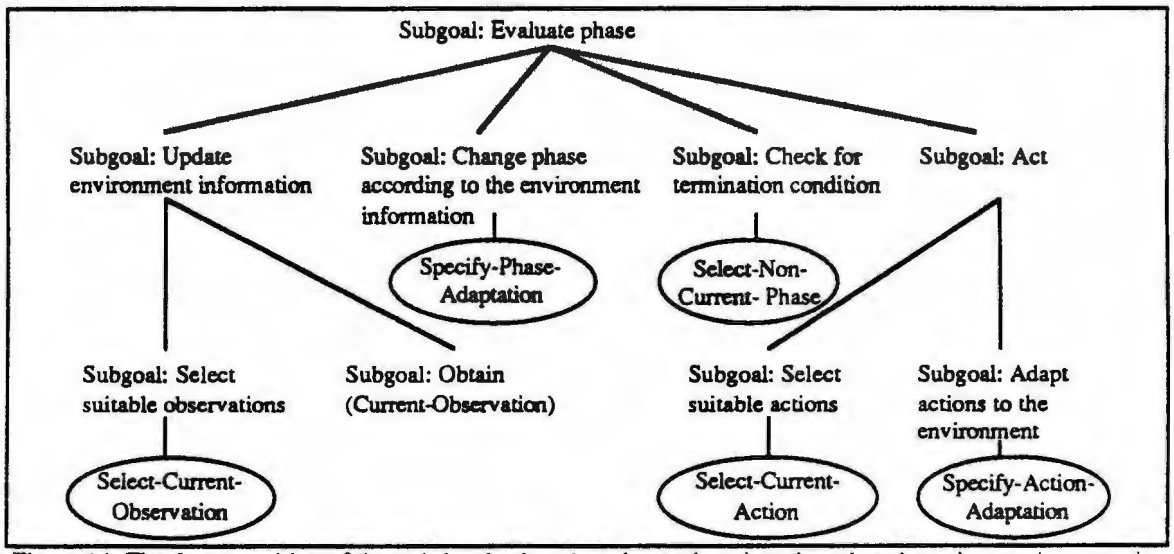


Figure 14: The decomposition of the task-level subgoal *evaluate phase* into the subgoals *update environment information, change phase according to the environment information, check for termination condition*, and *act*. Some of these subgoals must be decomposed further before they can be fulfilled by knowledge sources at the inference level.

The task structures of Tables 5 and 6 render the procedural aspects of the task decompositions of Figures 13 and 14. The task structure of Table 5 describes that while the knowledge source Select-Next-Phase can be evaluated successfully (i.e., as long as it inflicts some kind of change), the subtask Evaluate-Phase is executed. The task structure of Table 6 describes that while the meta-class Current-Phase refers to domain elements (i.e., as long as there are active therapies), the problem solver selects observations. If the knowledge source Select-Observation was successful, the selected observations are evaluated (Obtain Current-Observation), the data obtained are completed, first inferences are drawn (Abstract-Environment-Data), and necessary modifications to the phases (i.e., therapies) are specified. These may lead to an elimination of the phase (e.g., if the possible number of delays is exceeded). Subsequently, actions (i.e., drugs) are selected (Select-Current-Action), and their dosage is adapted to the knowledge about the environment (i.e., the laboratory-test results).

Note that *Obtain (Current-Observation)* is a task-layer statement, meaning that the system triggers an external query to obtain data.

The K-ONCOCIN task structures and the task decompositions are fixed; there is no need for dynamic modification of the decomposition. We therefore do not need a strategy layer to describe K-ONCOCIN.

Table 5: The top-level task structure captures the procedural aspects of the task decomposition of Figure 13. The task structure describes that while the knowledge source Select-Next-Phase can be evaluated successfully, the subtask Evaluate-Phase is executed.

Task: Refine skeletal plan
Goal: Refine a skeletal plan by selecting the next phase and evaluating it.
Task structure: WHILE Select-Next-Phase (Available-Phase + Environment-Data → Current-Phase) DO Evaluate-Phase

4.3. The Design Model

In a KADS design model (see Figure 2), the knowledge engineer describes which knowledge-representation techniques will be used to implement the elements of the conceptual model. The knowledge engineer's selection of techniques necessarily will be strongly influenced by the anticipated environment (expert-system shell). In our work, we have used BABYLON V 2.1 (Di Primio & Wittur, 1987; Guesgen, Junker & Voß, 1987; Christaller, Di Primio & Voß, 1989) to implement a functioning version of K-ONCOCIN. BABYLON V 2.1, LISP-based, integrates frames, PROLOG, rules, and constraints as knowledge-representation formalisms. K-ONCOCIN runs on a Macintosh and solves the task of cancer-chemotherapy administration for the Protocol 2083-1. A detailed description of the K-ONCOCIN design model will be published elsewhere (Linster, 1991). The same technique was used to implement a conceptual model for selection of clamping tools (Kühn, Linster & Schmidt, 1991).

Table 6: The task structure for the procedural aspects of the subtask decomposition of Figure 14. It describes that while the meta-class Current-Phase refers to domain elements, the problem solver selects observations (Select-Observation). In case this was successful, the selected observations are evaluated (Obtain (Current-Observation)), the data obtained are completed, first inferences are drawn (Abstract-Environment-Data), and necessary modifications to the phases (i.e., therapies) are specified. These may lead to an elimination of the phase. Subsequently, actions (i.e., drugs) are selected (Select-Current-Action), and their dosage is adapted to the knowledge about the environment (i.e., the laboratory-test results).

Task: Evaluate phase
Goal: Update the environment information, change the phase according to the environment information, check for termination condition, and act.
Task structure: WHILE NOT EMPTY Current-Phase DO IF Select-Current-Observation (Available-Observation + Current-Phase → Current-Observation) THEN Obtain (Current-Observation) Abstract-Environment-Data (Current-Observation → Environment-Data) Specify-Phase-Adaptation (Environment-Data → Current-Phase) Select-Noncurrent-Phase (Current-Phase → Obsolete-Phase) IF Select-Current-Action (Available-Action + Current-Phase → Current-Action) THEN Specify-Action-Adaptation (Environment-Data + Current-Action → Updated-Action)

In brief, we chose the frame and frame-instance representation mechanisms of BABYLON V 2.1 to encode concepts and attributes from the domain layer. This formalism allows for explicit preservation of the hierarchical relationships among the domain layer concepts shown in Figure 8. To implement the relations of the domain layer (e.g., *Drug-in-Chemotherapy*, *Test-Due*, and *Drug-Dose-Attenuation*), we used PROLOG clauses, as PROLOG allows both extensional description of relations through the enumeration of the tuples, and intentional description through characteristic predicates that are true for the tuples that satisfy the relations.

The inference-layer elements also are represented explicitly. Meta-classes are rendered as dynamic PROLOG predicates that return *true* for those domain-layer concepts that currently play the role in problem solving indicated by the meta-class. For example, we represent the meta-class *Current-Observation* in the design model by a predicate called *Current-Observation* that is satisfied by the domain-layer concepts such as *Test*, *Disease-Activity*, and so on, while these concepts play the role denoted by the meta-class. The knowledge sources are implemented as abstract rule schemata that can be related to the underlying domain relations. At run time, the problem solver instantiates the rule schemata with the necessary concepts and relations from the domain layer to form a set of executable forward-chaining rules.

The task layer of the conceptual model is implemented as a sequence of calls to the rule schemata that implement the knowledge sources.

Developers usually consider external requirements—such as speed, integration, and data management—in the design model. We developed K-ONCOCIN purely as an academic exercise however, so we did not develop a special interface, or a patient database.

5. DISCUSSION

We used KADS to remodel ONCOCIN as part of an experiment designed to help us understand the methodology; we did not turn to KADS to solve a real problem in knowledge engineering. K-ONCOCIN was developed not only from original medical protocol documents, but also from the data structures of ONCOCIN and OPAL. These previous computational models of the task of administering cancer therapy certainly provided us with biased preconceptions, and may have caused us to use KADS differently from the way it is used in more traditional knowledge-engineering situations. Nevertheless, our experience in developing K-ONCOCIN offers us an opportunity to make useful observations regarding the KADS approach.

5.1. Guidance for Knowledge Engineering

KADS is not a cookbook for building expert systems. In fact, KADS offers little guidance concerning how to construct a conceptual model and how to transform that conceptual model into a design model. The approach provides epistemological knowledge-structuring primitives (Brachman, 1979) for describing problem solving, consisting of concepts, structures, and relations on the domain layer; knowledge sources, meta-classes, and inference structures for the inference layer; and goals and tasks for the task layer. KADS does not give directives how to use these epistemological primitives in a given knowledge-engineering situation, but it does offer a concise terminological framework for the description of problem solving. In the domain-layer of our task, for example, KADS does not tell us whether we should view *chemotherapies* as complex structures that combine *consists-of* and *temporal* information to express the combinations of drugs and tests that make up a chemotherapy, or whether we should represent this information separately by specifying temporal relations, as we have done in the relations *Test-Due* and *Drug-in-Chemotherapy*.

By supplying typologies of meta-classes, knowledge sources (Table 2), and inference structures, KADS provides a useful set of terms and relations for modeling knowledge. This ontology, however, is far from being sufficient to categorize the universe of problem-solving behavior. For example, the library of interpretation models does not provide an inference structure that we could have used to model our task. The inference structure that was closest to our task is called *Hierarchical Design* (Breuker et al., 1987, p. 96). This structure, apart from the fact that it does not consider sequences of actions, does not provide the detailed modeling that our task required—instead it contains knowledge sources such as *Decompose*, *Specify*, *Design*, and *Aggregate*.

KADS does not provide guidance in the use of the ontology to categorize novel phenomena in a knowledge-engineering process. This ability would require directives for the representation of temporal, causal, factual, and all other kinds of knowledge on the domain layer. On the inference layer, such guidance would require a richer typology of inference-layer description elements and explicit criteria for the recognition of knowledge sources and meta-classes in verbal documents and transcripts. Given the current state of knowledge-acquisition research, however, the current KADS framework directs the knowledge engineer as well as possible in the context of an open-ended modeling environment.

KADS—rather than being prescriptive—offers a structured terminology that the knowledge engineer can use to create a conceptual model that may express a wide range of problem-solving methods in virtually any application domain. The transition between the conceptual model and the design model, however, requires the trained knowledge engineer to use skill in reconstructing the knowledge in the new, more restricted format. Although KADS thus offers considerable freedom in helping the developer to build a conceptual model, the technique does not assist the developer in overcoming the bias and information loss that can occur during the model-transformation phases.[2] Model transformation is still a poorly understood process that relies heavily on the knowledge engineer's intuition and programming skills. Nevertheless, making explicit the intermediate stages between the initial conceptual model and the ultimate program code can be a valuable step toward more principled approaches to knowledge engineering.

5.2. Other Knowledge-Acquisition Approaches

Automated knowledge-acquisition tools, such as SALT (Marcus, 1988; Marcus, 1989), KNACK (Klinker et al., 1989; Klinker, 1989), PROTÉGÉ/p-OPAL (Musen, 1989b), and OPAL (Musen et al., 1987), take a different view of the problem of knowledge engineering. Such tools provide a predefined problem-solving method, a set of generic actions and data-structures, and a specialized editor with which users can instantiate the generic actions and data structures with domain-specific knowledge (Figure 15).

The user thus must fit knowledge about the application to the problem-solving method and the latter's data structures. For example, the generic actions and data structures in SALT correspond to the schemata for fixes, constraints, and design extensions that are used in a problem-solving method known as *propose-and-revise* (Marcus, 1988). The knowledge that SALT can acquire must be expressed in only these terms. An extensive discussion of assumptions built into tools such as SALT has been presented elsewhere (Karbach, Linster & Voß, 1990).

---

[2]After development of the final system, knowledge engineers must continuously revise the documentation for the conceptual model to reflect changes made directly to the program code. Otherwise, the conceptual model will not be a valid documentation of the implemented system, and cannot be used as a basis for the ongoing maintenance.

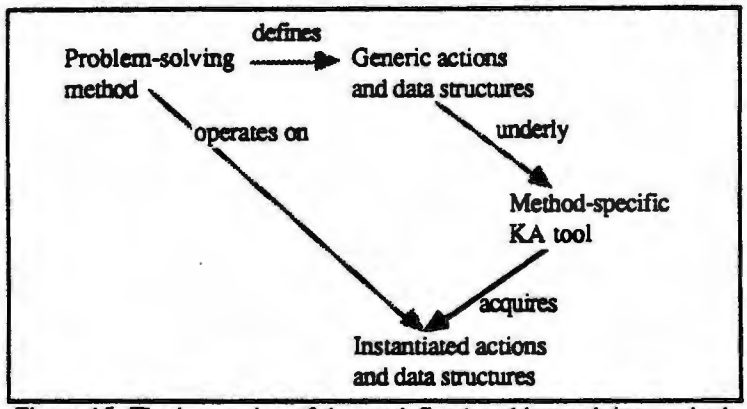


Figure 15: The interaction of the predefined problem-solving method, the generic actions and data-structures, and the specialized knowledge-acquisition editor in a SALT-like method-based tool (Source: adapted from Karbach, Linster & Voß, 1990, p. 295).

If a suitable knowledge-acquisition tool is available for an application, then the use of that tool may allow efficient knowledge entry and may provide helpful insights into the emerging knowledge base. The knowledge engineer, however, must know in advance whether a given tool is suitable for the application task at hand. KADS may be able to aid the knowledge engineer in this determination only if there exists a KADS model of the tools that can be compared with a KADS conceptual model of the application.

KADS conceptual models are application-specific, and thus contain all the knowledge needed to solve a particular task in a particular application domain. The upper layers of KADS conceptual models, however, are not application-specific. The task layer and the interpretation layer are described in generic terms. These two layers constitute a description of a domain-independent problem-solving method.

The domain layer of a conceptual model can be divided into (1) the type declarations, and (2) the instance definitions. This class–instance distinction applies to the relations as well, which first are declared and then are instantiated with n-tuples. In the domain layer of K-ONCOCIN (see Figure 8), the type declarations include *Chemotherapy*, *Radiotherapy*, *Drug*, *Radiation*, *Patient*, and *Test*. The corresponding instances include *CcOM*, *CAVP*, and so on.

Thus, within KADS conceptual models, we can distinguish among the model of the problem solving-method (the strategy, task, and interpretation layers of the model), the model of the task (the problem-solving model with the type declarations from the domain layer), and the model of the application (the conceptual model as a whole). The KADS approach thus could serve as an initial step in the development of knowledge-acquisition tools like SALT (which are based on models of problem-solving methods), and tools like OPAL (which are based on models of tasks), in addition to aiding the conceptual design of complete expert-system applications.

5.3. Lessons from K-ONCOCIN

Based on our development of K-ONCOCIN, we can identify several issues that researchers should address as they work to extend and refine the KADS method.

The library of existing interpretation models must be extended to cover a wider range of problems. The elements of the library need to be described in much greater detail. To select an interpretation model, the knowledge engineer needs to refer to a similar system that was built to solve a similar task, using the approach that Bennett (1985) adopted in ROGET. Novice users of the KADS approach, however, need to know more than simply which interpretation models were used previously for modeling particular tasks; novices need extensive case studies that they can use as exemplars for the application of the KADS methodology. One of our goals in developing K-ONCOCIN was to document such an exemplar.

Currently, the interpretation models of the library describe neither which kind of domain-layer phenomena they explain nor the knowledge that they require. It is thus difficult to recognize whether a complex interpretation model, such as *planning*, is applicable to a task at hand, or whether the knowledge engineer should develop a new one. This description can be informal and does not need to be stated as data structures. For example, if a knowledge engineer wants to apply the interpretation model of K-ONCOCIN, then the task must be solvable by a sequence of actions that are executed in a predefined order, and the fine-tuning of the sequence and of the actions must depend on tests or observations. These building-blocks must be present in the knowledge that was elicited. Otherwise, our model is not applicable.

Currently, the knowledge engineer manually transforms the conceptual model into a design model, which she then transforms into an operational system. Every change that is made to the conceptual model requires transformation of the corresponding documentation and, ultimately, transformation of the operational program code. Automated tools are needed urgently, so that the details of the domain-layer knowledge can be maintained and adapted by application specialists on site after delivery of the final expert system. This requirement calls for editors like p-OPAL that allow their users to view the knowledge base in familiar terms and to change the domain knowledge within the framework defined by the KADS conceptual model.

ACKNOWLEDGMENTS

We thank Lyn Dupré, Werner Karbach, Otto Kühn, Karin Lagrange, Guus Schreiber, Nigel Shadbolt, Angi Voß, Bob Wielinga, and Brian Woodward for comments on earlier versions of this paper. This work was supported in

part by grant LM05157 from the U.S. National Library of Medecine, and by grant HS06330 from the U.S. Agency for Health Care Policy and Research.

REFERENCES

Alexander, J., Freyling, M., Shulman, S., Rehfuss, S. and Messick, S. Ontological analysis: An ongoing experiment. *International Journal of Man–Machine Studies* **26** (4) (1987) 473–485.

Anjewierden, A. and Wielemaker, J. Shelley: Computer aided knowledge engineering. In: Wielinga, B., Boose, J., Gaines, B., Schreiber, G. and van Someren, M. (eds.), *Proceedings of EKAW90*. IOS Press, Amsterdam, 1990, 41–59.

Bennett, J.S. ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning* **1** (1) (1985), 49–74.

Boose, J.H., Shema, D.B. and Bradshaw, J.M. Recent progress in AQUINAS: A knowledge acquisition workbench. Knowledge Acquisition **1** (2) (1989), 185–214.

Brachman, R.J. On the epistemological status of semantic networks. In: Findler, N.V. (ed.), *Associative Networks: Representation and Use of Knowledge by Computers*. Academic Press, London, 1979, 3–50.

Bredeweg, B., de Greef, P., Schreiber, G., Terpstra, P., Wielinga, B., Brunet, E., Simonin, N. and Wllyn, A. A KADS approach to KBS design. KADS Technical Report B6, ESPRIT Project P1098, University of Amsterdam, Amsterdam, 1989.

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billaut, J.P., Davoodi, M. and Hayward, S. Model-driven knowledge acquisition: Interpretation models. KADS Technical Report A1, ESPRIT Project P1098, University of Amsterdam, Amsterdam, 1987.

Breuker, J. and Wielinga, B. Models of expertise in knowledge acquisition. In: Guida, G. and Tasso, C. (eds.), *Topics in Expert -System Design, Methodologies and Tools*. North-Holland, Amsterdam, 1989, 265–295.

Carlson, R.W., Sikic, B.I., Gandara, B.R., Cadman, E.C. and Friend, N. A phase III trial of seven-drug versus three-drug chemotherapy regimens with or without prophylactic cranial irradiation for undifferentiated small cell anaplastic lung cancer: extensive disease. Technical Report 2083-1, Northern California Oncology Group, Palo Alto, CA, 1986.

Chandrasekaran, B. Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples. *The Knowledge Engineering Review* **3** (3) (1988), 183–210.

Christaller, T., di Primio, F. and Voß, A. (eds.), *Die KI-Werkbank BABYLON*. Addison-Wesley, Bonn, 1989.

Clancey, W. Heuristic classification. Artificial Intelligence **27** (1985), 289–350.

Combs, D.M. The OPAL knowledge acquisition tool: Functional specification. Internal Memo, Medical Computer Science Group, Knowledge Systems Laboratory, Stanford University, Palo Alto, CA, 1988.

Gaines, B.R. and Shaw, M.L.G. Cognitive and logical foundations of knowledge acquisition. In: Gaines, B.R. and Boose, J.H. (eds.), *Proceedings of KAW90*. University of Calgary, Calgary, 1990, 9/1–9/25

Gruber, T.R. A method for acquiring strategic knowledge. *Knowledge Acquisition* **1** (3) (1989), 255–277.

Guesgen, H.W., Junker, U. and Voß, A. Constraints in a hybrid knowledge representation system. In: McDermott, J. (ed.), *Proceedings of IJCAI87*, Morgan Kaufmann, Los Altos, 1987, 30–33.

Karbach, W., Linster, M. and Voß, A. Models of problem-solving in knowledge based systems: their focus on knowledge acquisition. *Knowledge Acquisition* **2** (4) (1990), 279–300.

Klinker, G., Boyd, C., Dong, D., Maiman, J., McDermott, J. and Schnellbach, R. Building expert systems with KNACK. *Knowledge Acquisition* **1** (3) (1989), 299–320.

Klinker, G. A framework for knowledge acquisition. In: Boose, J.H., Gaines, B.R. and Ganascia, J.G. (eds.) *Proceedings of EKAW89*, Université Paris-Sud, Paris, 1989, 102–116.

Kühn, O., Linster, M. and Schmidt, G. Clamping, COKAM, KADS and OMOS. In: Smeed, D., Linster, M., Boose, J.H. and Gaines, B.R. (eds.), *Proceedings of EKAW91*, Springer, Berlin Heidelberg New York, in press.

Linster, M. Implementing K-ONCOCIN with OMOS. Technical report of GMD, GMD, St. Augustin, in press.

Marcus, S. A knowledge acquisition tool for propose-and-revise systems. In: Marcus, S. (Ed.) Automating knowledge acquisition for expert systems. Kluwer Academic Publishers, Boston, 1988, 81–124

Marcus, S. Understanding decision ordering from a piece-meal collection of knowledge. *Knowledge Acquisition* 1 (3) (1989) 279–298.

McDermott, J. Preliminary steps toward a taxonomy of problem-solving methods. In: Marcus S. (ed.), *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Press, Boston, 1988, 225–256.

Musen, M.A., Fagan, L.M., Combs, D.M. and Shortliffe, E.H. Use of a domain-model to drive an interactive knowledge-editing tool. *International Journal of Man–Machine Studies* **26** (1987) 105–121.

Musen, M.A. Automated support for building and extending expert models. *Machine Learning, Special Issue on Knowledge Acquisition* 4, (3, 4) (1989) 347–376.

Musen, M.A. *Automated Generation of Model-Based Knowledge Acquisition Tools*. Pitman, London, 1989 (Research Notes in Artificial Intelligence).

Musen, M.A. Conceptual models of interactive knowledge acquisition tools. *Knowledge Acquisition* 1 (1) (1989) 73–88.

Newell, A. The Knowledge Level. *Artificial Intelligence* 18 (1982) 87–127.

Di Primio, F. and Wittur, K. BABYLON: A meta-interpretation model for handling mixed knowledge representations. In: *Proceedings of the seventh International Workshop on Expert Systems and Their Applications*, Avignon, 1987, 821–833.

Punch, W.F. and Chandrasekaran, B. An investigation of the roles of problem-solving methods in diagnosis. In: *Proceedings of the tenth International Workshop on Expert Systems and Their Applications*, Avignon, 1990, 25–36.

Schachter-Radig, M.J. and Wermser, D. A sales assistant for chemical measurement equipment. In: Kodratoff, Y. (ed.), *Proceedings of ECAI88*, Pitman, Munich, 1988, 191–193.

Schreiber, G., Breuker, J., Bredeweg, B. and Wielinga, B. Modeling in KBS development. In: Boose, J.H., Gaines, B.R. and Linster, M. (eds.), *Proceedings of EKAW88*. GMD, St. Augustin, 1988, 7/1–7/15.

Shaw, M.L.G. and Gaines, B.R. KITTEN: Knowledge initiation and transfer tools for experts and novices. *International Journal of Man-Machine Studies* 27 (3) (1987), 251–280.

Shortliffe, E., Scott, A., Bischoff, M. and van Melleand C. Jacobs, W. ONCOCIN: An expert system for oncology protocol management. In: *Proceedings of IJCAI81*, Vancouver, 1981, 876–881.

Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J.C., Shortliffe, E.H. and Fagan, L.M. Episodic skeletal-plan refinement based on temporal data. *Communications of the ACM* 32 (12) (1989) 1439–1455.

Voß, A., Karbach, W., Drouven, U. and Lorek, D. Competence assessment in configuration tasks. In: Aiello, L.C. (ed.) *Proceedings of ECAI90*. Pitman, London, 1990, 676–681 (DIGITAL European AI Research Paper Award).

Wielinga, B. and Breuker, J. Models of expertise. In: *Proceedings of ECAI86*. Brighton, 1986, 306–318.

Wielinga, B., Akkermanns, H., Schreiber, G. and Balder, J. A knowledge acquisition perspective on knowledge level models. In: Boose, J.H. and Gaines, B.R. (eds.), *Proceedings of KAW89*. University of Calgary, Calgary, 1989, 36/1–36/22.

Wielinga, B., Schreiber, G., and Breuker, J. KADS: A modeling approach to knowledge engineering. KADS-II Technical. Report T1.1, ESPRIT Project P5248, University of Amsterdam, Amsterdam, 1991.

More Data on Usable and Reusable Programming Constructs

David Marques, Georg Klinker, Geoffroy Dallemagne,
Patrice Gautier, John McDermott, and David Tung

Digital Equipment Corporation
111 Locke Drive
Marlboro, MA 01752

Abstract

To make programming easier, we are designing and implementing a framework, the Easy Programming framework, exploiting programming constructs, called mechanisms, that are both usable and reusable. A mechanism is usable if it can be used to automate an application task by someone who understands and performs that task, but who does not necessarily know how to program. A mechanism is reusable if it can be employed for several tasks and domains. Some first results were presented last year (Klinker, Bhola, Dallemagne, Marques, McDermott, 1991). Since then, we have used our framework to create more application programs and have refined our understanding of mechanisms. This paper reports the status of our ongoing research project and presents more data addressing our claims.

1 Introduction

Our goal is to make effective programming easier. Effective programming is the construction of application programs that are useful. Easier means programming takes less time and effort, and easier means people with less programming skill -- or none -- can create useful programs. We accomplish this by building programs from reused, customizable programming constructs we call mechanisms.

Programming is a matter of configuring and instantiating programming constructs to create a working application. At one extreme, assembly languages provide tiny, generic, programming constructs (eg, move, branch-if-equal). Only a very experienced programmer can create a configuration of those constructs that map onto an application task. At the other extreme, some monolithic inference engines that are essentially one large, highly-specialized mechanism (eg, a report generator) present no configuration problem, but do not allow much instantiation as they are already very specialized. The problem is that the applicability (application scope) of such mechanisms is limited, and the difficulty for

many of the specialized inference engines can become one of determining when the mechanism is appropriate.

We can use these extremes as endpoints in describing approaches to programming in three dimensions: user scope, application scope, and functional scope. Figure 1-1 is a graphic representation of this description. For example, traditional programming languages (lower right) use small programming constructs (small functional scope) and tend to have wide applicability (large application scope) but small usability (small user scope).

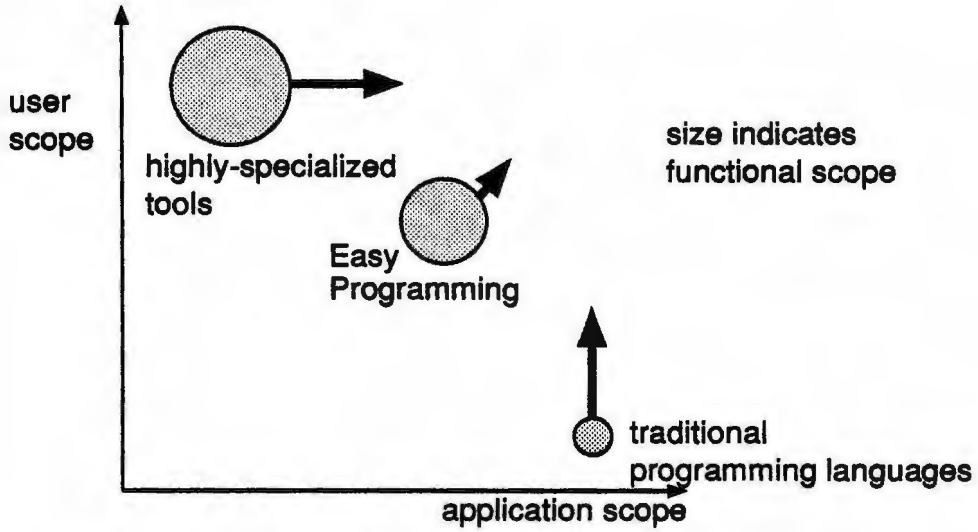


Figure 1-1 Approaches for Making Programming Easier

The goal of making programming easier is shared by many efforts, but the approaches vary. One approach to making programming easier (the vertical arrow in Figure 1-1) is to start from programming languages and create programming constructs that are more easily understood than traditional language constructs and therefore more easily configured. One successful approach is that of (Rich, Waters, 1990) who use mechanisms (*cliches* in their terminology) that are functionally larger than those of traditional programming languages, but still fairly small and have broad, general applicability. It takes many of their mechanisms to make an application. Thus, the task of determining how to combine the reusable parts into a working program is easier than with traditional languages but still requires significant programming skill. Similar work by others (Freeman, 1987; Krueger, 1989) is aimed primarily at helping experienced programmers. The suggested programming constructs are small in functional scope and user scope but large in application scope.

Other approaches (the horizontal arrow at the top of Figure 1-1) aim to help "non-programmers" with tools to act as interpreters between complex pre-constructed programming shells and an application developer. These automated knowledge acquisition shells can build a limited set of similar applications (Bennett, 1985; Birmingham, 1988; Boose, 1984; Clancey, 1983; Davis, 1979; Eshelman, 1988; Klinker, 1988; Marcus, 1988; Musen, 1989). Here researchers try to build programming constructs maximizing user scope -- a person who understands and performs a task can, without traditional programming experience, use the tools to create an application. They do this by having only one or perhaps a very few mechanisms, reducing or eliminating the need for the application builder to configure the mechanisms. This relatively large functional scope allows the use of knowledge acquisition tools that are focused, providing help to the user, thereby decreasing the programming skills needed in the user for instantiating the mechanisms. These shells or inference engines, however, have a narrower applicability than traditional programming languages -- that is, they can be used for a much smaller range of

problems than tools built by those using languages with many building blocks. The suggested programming constructs are large in user scope and functional scope but small in application scope.

We aim to build mechanisms that have a larger application scope (ie, can be used to build a larger range of applications) than the monolithic inference engines, but also have a larger user scope (ie, can be used by developers with less programming skill) than traditional programming languages. Our approach is to build mechanisms that are large enough (have large functional scope) to allow use of powerful knowledge acquisition tools for instantiating them, yet small enough to allow recombination, increasing their applicability.

We are building a library of mechanisms and associated knowledge acquisition tools as well as a framework configuring and exploiting them. The framework consists of tools that interpret between the application developer (not necessarily a programmer) and the mechanisms and that configure the mechanisms for the developer.

Section 2 very briefly presents our framework. A detailed description can be found in (Marques, 1991). Section 3 describes our empirical approach towards a theory of mechanisms. Section 4 presents the experiments -- the method, the applications and the mechanisms. Section 5 presents the results of the experiments. Finally, section 6 concludes with an outlook to related work.

2 A Framework for Making Programming Easier

The main problems in building application programs arise from the nature of real tasks. Real tasks are messy and filled with details that make them unique. These uniquenesses are often the most important parts to the end-user. The messiness makes it difficult to abstract from the task a description of the computational process for performing the task. The people who perform the task are caught up in the details of the task and find it difficult to design an abstract method for accomplishing that task. In programming, this abstract method is the control structure, and the first problem with making programming easy is mapping the features of the task to the appropriate control structures. Method abstraction is addressed by our work on the tool Spark.

The messiness of tasks also makes it difficult to gather expertise, full of exceptions and variations, and translate it into a form that can be used by an abstract control structure. This knowledge acquisition problem is addressed by our work on the tool Burn.

The messiness of tasks makes it inevitable that early versions of application programs will not cover the task well. In fact, real tasks change continuously over time. The population of users changes, the kinds of information desired change, and the goals of the task change. Sometimes these changes are subtle, but they are inevitable. Thus any application built must, as a primary criterion, be easy to modify, extend, and even to significantly change. Our approach to this problem is successive approximation through iteration and correction of a working program. We hope thereby to build application programs that do real work. Successive iteration by changing the way the program works is addressed by our work with the tool FireFighter.

Figure 2-1 illustrates our approach.

1. Spark helps the developer map a task to a fairly generic pre-defined task model, and helps the developer customize that generic task model to more accurately reflect the developer's task.
2. Associated with the task model are sets of pre-defined chunks of control structure that we call mechanisms.
3. The mapping is refined by Spark to yield a configuration of mechanisms that we call a method.
4. Burn takes the configuration of mechanisms and builds a configuration of knowledge acquisition tools.

5. These knowledge acquisition tools interview the developer to gather knowledge and translate it into useful representations resulting in an application program.
6. FireFighter takes the applications that are built by Burn and interviews the developer to identify errors in method or the coverage of knowledge acquired. FireFighter then iterates with Spark and Burn to refine, modify, and enhance the application program.

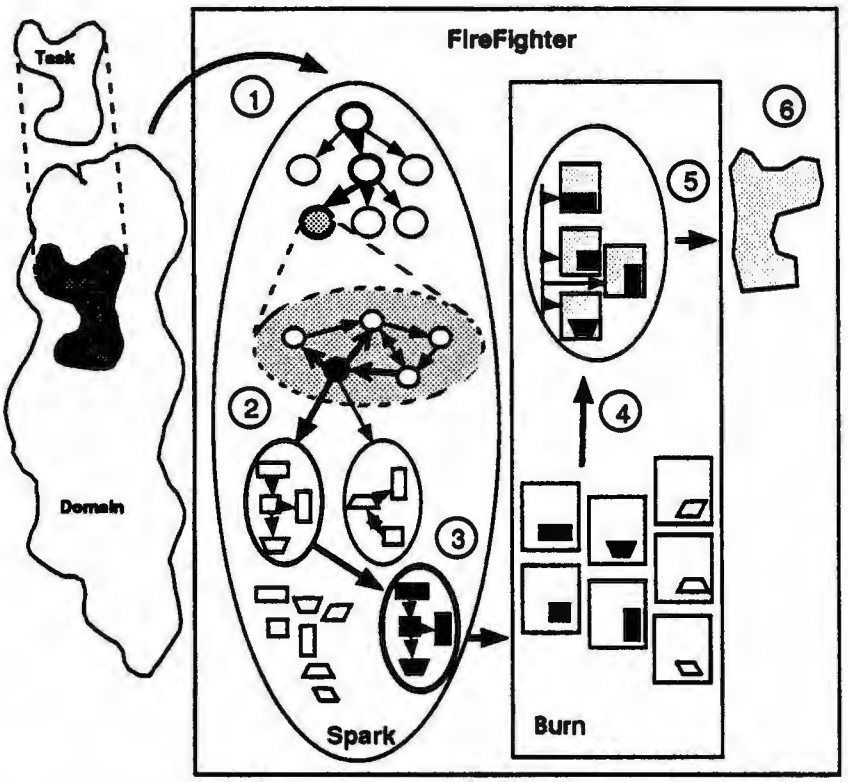


Figure 2-1 An Integrated Framework for Making Programming Easier

3 An Empirical Approach Towards Usable and Reusable Programming Constructs

We claim that our approach will maximize both user scope and application scope by optimizing functional scope. These claims make testable predictions:

- our mechanisms can be used to create useful applications;
- our mechanisms can be used in several different applications, so that the number of mechanisms needed to cover a large number of tasks is not unmanageably large (application scope);
- our tools (Spark, Burn, Firefighter) enable people with little or no programming skill to build useful applications (user scope);
- the number of mechanisms needed to cover an application is small enough to minimize or eliminate the necessity for application developers to configure mechanisms (functional scope).

The above hypotheses are only meaningful if we tie them to the effort involved in creating applications. It is easy to assert that, if those hypotheses are verified, our framework reduces the effort to create applications. However, we need to present evidence to support the following claim:

- the effort to build an application with our tools is less than without them.

To gather evidence about these hypotheses we are taking an empirical approach -- we build applications without and with mechanisms and attempt to reuse mechanisms when appropriate.

4 The Experiments

There are 4 stages for each application in our experiments. First, we build the application ourselves and present these applications to end-users who actually perform the task. If the users determine that the application program helps, we know that the application is actually useful. Second, if the program proves useful, we then write detailed descriptions of the task, and ask people of various programming experience to build a small version of the application with our tools to determine if our framework can be used to build the application. This stage might involve refining any component of our framework (mechanism, methodology, and/or tool) if a component cannot accommodate the new application. For the third stage, we ask people who perform the task well to create a full-scale application using our tools. Finally, we ask experienced programmers to build the full-scale application without using our framework and compare the time and effort that takes with the time and effort of those who used our tools.

From these efforts we derive the characteristics of mechanisms that are appropriate for maximizing application and user scope by optimizing functional scope. The next two sections describe the applications and mechanisms that we have used thus far and present data collected about our predictions.

4.1 The Applications

Qualifier: The task is to estimate the amounts of various hardware resources that are needed to satisfy the computational demands of a prospective client's organization, to the degree of accuracy necessary to qualify the sales opportunity.

- Desired results: a list of computer equipment which includes the capacities and number of CPU's, workstations, main memory, tape drives, disk storage, printers and terminals.
- Data available: a superficial description of the client's organization including the number of personnel in each job classification (such as traders, programmers, managers and secretaries), the industry in which the organization exists (eg, finance, publishing), and special computational needs such as real-time, access to databases, high speed graphic processing.
- Expertise available: knowledge of successful existing sizings, knowledge of how to discriminate among existing sizings, and knowledge of how to extrapolate from an already sized case to a solution for the current case.

SWSizer: The task is to select software tools necessary for a client's computer needs, and to estimate the number of licenses needed.

- Desired results: a list of software tools appropriate to a client's needs, and an estimate of the number of licenses for each software product.
- Data available: a rough description of the client's organization, the number of personnel in each job classification (such as traders, programmers, managers and secretaries), the type of software needed by the workers (such as programming languages or business graphics), special needs such as reconfigurable user interfaces; a database of software product descriptions.

- Expertise available: knowledge of existing orderable software, knowledge of constraints pertaining to the selection of specific software, knowledge of the relationship between number of users and number of licenses needed (for example, 10 programmers need one site license, part number QU3998-HZ)

Candlestick: The task is to estimate the amounts of various host and desktop hardware resources that are needed to satisfy the computational demands of a prospective client's organization, and the approximate connectivity among the CPUs.

- Desired results: a list of computer equipment which includes the capacities and number of CPU's, main memory, tape drives, disk storage, printers and terminals, workstations, workstation memory, workstation disk storage, and a diagram of how the host and its desktop devices are connected.
- Data available: a superficial description of the client's organization including the number of personnel in each job classification (such as engineers, scientists, programmers, managers and secretaries), the industry in which the organization exists (eg, automotive, publishing), and special computational needs such as real-time, access to databases, high speed graphic processing.
- Expertise available: knowledge of successful existing sizings, knowledge of how to discriminate among existing sizings.

PWSizer: The task is to determine the amounts of various hardware resources that are needed to satisfy the computational demands of a prospective client for PriceWatch (a specialized trader's workstation software), with an accuracy acceptable for preliminary proposals.

- Desired results: a list of computer equipment which includes the capacities and number of CPU's, workstations, main memory, tape drives, disk storage, printers and terminals.
- Data available: a detailed description of the client's organization including the number of personnel in each job classification (such as traders, managers and sales people), the market specialization of the organization, what instruments are traded, and special computational needs such as on-line histories, electronic data sources (for example, Reuters' RTDF and Marketfeed 2000, and Knight-Ridder), and methods of using PriceWatch such as page parsing, composite pages, and number of items per page.
- Expertise available: knowledge of successful existing sizings, knowledge of how to discriminate among existing sizings.

SWSelector: The task is to recommend application software for a client's organization, given a detailed description of the type of organization and their application needs. The first version covered only those organizations in the finance industry, in capital markets.

- Desired results: a list suggested application software products; a list of needs met and not met by each suggested software product.
- Data available: a detailed description of the client's organization, the activities performed within the client's organization, such as which capital markets the organization deals in, which instruments are watched or traded, which market analysis is performed by analysts.
- Expertise available: knowledge of existing selections, knowledge of constraints pertaining to the selection of specific software.

Layout: The task is to take a model of a user's data needs, create a graphical representation of the model in the form of rectangular tiles, and display that representation on a workstation screen.

- Desired results: a tiled workstation screen corresponding to an individual's data needs based on his view of his organization (for example, UIL code).
- Data available: a model corresponding to the user's data needs based on his view of his organization (for example, line item information with details about part number, number of items sold,

sales prize, customer, etc), and a description of a workstation's screen (for example, height and length).

- Expertise available: a description of how to access the model of a user's data needs, (for example, the location of a file containing the model), heuristics for placing tiles on the screen (for example, in a hierachical relationship, larger tiles represent aggregations of smaller tiles), topological knowledge to constrain tile placement on the screen (for example, the length of a tile cannot exceed the length of the workstation), and heuristics for refining tile placements (for example, if there is not enough space left to place a tile next to its siblings, start a new row), and knowledge of how to present the representation on a workstation's screen (for example, UIL code).

Retriever: The task is to generate code aggregating data from multiple databases into an individual's data needs based on his view of his organization.

- Desired results: code to aggregate data corresponding to an individual's data needs based on his view of his organization (for example, SQL code).
- Data available: a model describing how to aggregate the data in the databases to an individual's data needs based on his view of his organization (for example, individual customer data are aggregated to regional customer data via zip codes).
- Expertise available: a description of how to access the model of how to aggregate the data in the databases to a user's data needs, (for example, the location of a file containing the model), heuristics for aggregating the data (for example, data representing sales can be added to result in the overall sales), and knowledge of how to generate code (for example, SQL code).

Scheduler: The task is to take a list of orders and their due dates, create a schedule describing the manufacture of the orders, and present the schedule on a workstation screen.

- Desired results: a schedule that satisfies the relevant constraints (for example, start and end times for each operation to be performed) presented on a workstation screen.
- Data available: the orders and their due dates (for example, order "P1" is due on 7/9/92).
- Expertise available: a description of how to access order information (for example, the location of a file containing the orders); expertise describing the orders (for example, type of orders), resources (for example, worker, machine, material), operations (for example, the "lay-up" operation precedes the "bag" operation), and expertise constraining how specific resources can be associated with particular orders (for example, the "bond mold" resource "BM3" can only be used to manufacture orders of type "rotor"); the format of the table presenting the schedule (for example, columns are "start time", "end time"; rows are "order" and "operation")

Spark: The task is to determine which set of computational mechanisms can collectively solve a given application task.

- Desired results: a set of computational mechanisms appropriate for performing the specified task (for example, the mechanisms "Translate_01", "Eliminate_01", and "Translate_02" are appropriate to solve Spark's task).
- Data available: a set of computational mechanisms (for example, the mechanisms described in this section), the questions and possible answers used for describing a task (for example, the set of questions and possible answers associated which each mechanism that confirm or reject the relevance of a mechanism's desired results, available data, and available expertise).
- Expertise available: heuristics mapping the possible answers to the computational mechanisms (for example, if the expertise necessary for a mechanism cannot be provided, that mechanism is rejected).

PC-Config: The task is to create a complete and valid configuration of PC devices and associated software, and estimate the number of servers and printers required for an organization, given a rough description of the users of the PC devices. The task is also to make many small changes to the suggested configuration still guaranteeing a valid configuration with each change.

- Desired results: a detailed list of each desktop device (PC) recommended, including every configured part (for example, quantity 5 of DECstation 425 with 16 MB memory, 120 MB disk, SCSI interface, 6 expansion ports, color monitor with Super VGA graphics controller, standard keyboard -- and associated part numbers); a list of compatible software for each desktop and for the servers (for example, Lotus 1-2-3 for DOS, Pathworks for the server); an estimate of the number of servers and shared printers.
- Data available: a rough description of the client's organization, the number of personnel in each job classification (such as traders, programmers, managers and secretaries), the type of software needed by the workers (such as programming languages or business graphics), special needs such as reconfigurable user interfaces; a database of software product descriptions.
- Expertise available: a set of example valid configurations and the most appropriate users; a set of all orderable parts and configuration constraints (for example, a SCSI-type drive requires a SCSI controller).

4.2 The Mechanisms and Associated Knowledge-Acquisition Tools

A mechanism is a code skeleton that is instantiated by its associated knowledge-acquisition tool. Therefore, it can be described by the data that need to be available to it, the results it produces, and the expertise that must be available to parameterize the mechanism. The available expertise is what is acquired by a mechanism's knowledge-acquisition tool.

Dialog-mgr: The task is to create a series of interactive queries and displays.

- Desired results: a series of queries or displays (for example, a table to be filled in by a user, an image to be displayed on the screen).
- Data available: a description of an object or a set of objects (for example, the question to be asked, the possible answers, a default answer, etc.).
- Expertise available: heuristics defining the format used for a query or display (for example, if a user must select courses he wants to take from a pre-defined list of possible courses, create a multiple choice menu)

Select: The task is to select a specific element from a tree, list, or set structure of possible elements based on the characteristics of those structures.

- Desired results: a selected element (for example, a tile to be placed on a workstation's screen).
- Data available: the data organized by the relevant structure (for example, the tiles to be placed on a workstation's screen).
- Expertise available: expertise on how to select the next element (for example, place the tiles on a workstation's screen by traversing the tree structure representing them in a depth first manner).

Classify: The task is to locate one or more sets of data in a database, given a list of characteristics, based on a qualitative best-match algorithm.

- Desired results: a list of sets of data, equally matched with a set of indexing criteria (for example, the best diagnosis given a set of symptoms, or the set of best solutions -- each arbitrarily complex -- for a given problem description)

- Data available: a description of a problem in question/answer or attribute/value format (for example, "Which industry?" -- "Automotive", or "spark-plug-condition" -- "blackened" & "not worn") where the possible answers are pre-enumerated.
- Expertise available: a description of solutions in any format but with associated indexes in question/answer or attribute/value format (for example, C language license with ULTRIX operating system, VAX4000-class host, DECstation 5000-class workstations, included diagram -- matched with "User Environment" -- "Engineering" and many other indices).

Translate_01: The task is to read data from a disk and translate them from one format into another pre-defined, internal representation that mechanisms can operate upon.

- Desired results: data in a format that mechanisms can use (for example, list, set, and tree structures).
- Data available: a file on disk whose records either define a new data structure or define the individual data (for example, the start or end of a list, an element of a list, the values of list attributes etc).
- Expertise available: the location of the file on the disk, and the format of the data records on the disk (for example, the definition of the attributes characterizing the elements of a list).

Translate_02: The task is to translate data from a pre-defined, internal representation that mechanisms can operate upon into another desired format and write those to a disk.

- Desired results: a file on a disk whose records correspond to the desired format (for example, UIL code, or SQL code).
- Data available: data in a format that mechanisms can use (for example, list, set, and tree structures).
- Expertise available: the location of the file on the disk, and the name of the representation to be produced (for example, UIL or SQL).

Transform_01: The task is to estimate a set of numbers from an example set and corresponding linear extrapolation rates for each number.

- Desired results: a set of numbers appropriately sized for a corresponding numeric description (for example, for 5 programmers, 4 secretaries, 3 scientists, use a host with 21 MIPS, 600 MB disk, 32 MB memory, 8 workstations and 4 terminals).
- Data available: numeric description of the current problem (for example, 5 programmers, 4 secretaries, 3 scientists).
- Expertise available: a reference example of problem, solution, and linear extrapolation rate (for example, start with 15 MIPS for 5 programmers, add 1.5 MIPS for each new programmer).

Transform_02: The task is to create a description of the spatial interdependencies for a set of rectangles that corresponds to a user's data needs, based on his view of his organization.

- Desired results: a description of the spatial interdependencies for a set of rectangles (for example, each rectangle must be characterized by its length, height, label, position on a workstation screen, etc).
- Data available: a model corresponding to a user's data needs, based on his view of his organization (for example, line item information with details about part number, number of items sold, sales prize, customer, etc), and a description of a workstation's screen (for example, height and length).
- Expertise available: heuristics for placing tiles on the screen (for example, in a hierachical relationship, larger tiles represent aggregations of smaller tiles), topological knowledge to constrain tile placement on the screen (for example, the length of a tile cannot exceed the length of the

workstation), and heuristics for refining tile placements (for example, if there is not enough space left to place a tile next to its siblings, start a new row).

Compare_01: The task is to create a list of descriptions, rank-ordered in numeric difference from a reference description, based on numeric parameters.

- Desired results: a rank-ordered list of descriptions (for example, description_1 is closest, followed by description_75, then description_16).
- Data available: a set of descriptions and a reference description (for example, description_1 has 2 programmers, 4 scientists, 2 secretaries; description_75 has 3 programmers, 6 scientists, 1 secretary; reference has 3 programmers, 9 scientists, 2 secretaries).
- Expertise available: formula for calculating Euclidean distance between two sets of vectors.

Compare_02: The task is to cause other mechanisms to create two solutions and then compare them and report how much they differ.

- Desired results: a description of how two solutions differ (for example, "These two solutions for MIPS disagree by 35%, which indicates that you should review both reference examples in more detail.").
- Data available: descriptions of two different solutions.
- Expertise available: what message to give for what level of disagreement (for example, 0-25% "Two different reference examples yielded very similar solutions").

Eliminate: The task is to select a subset from a set structure of possible elements based on answers to questions.

- Desired results: a selected subset (for example, a set of computational mechanisms appropriate for performing a specified task).
- Data available: the data organized by the set structure (for example, a set of computational mechanisms), and the questions and possible answers (for example, a description of a task).
- Expertise available: heuristics mapping the possible answers to elements of the set structure (for example, heuristics for determining which of the set of computational mechanisms are needed to automate a task described by the possible answers.

Monitor: The task is to execute mechanisms in an order specified by a user.

- Desired results: a mechanism to be executed.
- Data available: a list of mechanisms that have been executed within the current run of the application program.
- Expertise available: a list of mechanisms that can be executed, (for example, "Translate_01", "Schedule_01", and "Present_01"), menu choices corresponding to those mechanisms (for example, the menu choice "Get orders" corresponds to the mechanism "Translate_01").

Present: The task is to present a table on a workstation screen.

- Desired results: a table presented on a workstation screen (for example, a schedule listing -- for each order -- operations to be performed, their start and end times; a table detailing utilization for each resource and time interval).
- Data available: the data to be presented (for example, the time information for the scheduling data).
- Expertise available: the format of the table (for example, columns are "start time", "end time"; rows are "order" and "operation").

Schedule: The task is to use a heuristic approach to associate orders with operations and resources, such, that given constraints are satisfied.

- Desired results: a schedule that satisfies the relevant constraints (for example, start and end times for each operation to be performed).
- Data available: the orders and their due dates (for example, order "P1" is due on 7/9/92).
- Expertise available: expertise describing the orders (for example, type of orders), resources (for example, worker, machine, material), operations (for example, the "lay-up" operation precedes the "bag" operation), and expertise constraining how specific resources can be associated with particular orders (for example, the "bond mold" resource "BM3" can only be used to manufacture orders of type "rotor").

Constrain: The task is to satisfy a new constraint imposed on an already valid solution, and either produce a new, valid solution or indicate why it cannot be done.

- Desired results: a new valid solution incorporating the new constraint (for example, changing the monitor from monochrome to color, and automatically including the appropriate cable and color boards if necessary).
- Data available: the new constraint (for example, change monitor to color).
- Expertise available: a list of all constraints for combining all parts (for example, color monitor needs cable YY-M3, or software XX needs 1 MB memory and 30 MB disk).

5 Results

The following data are preliminary results from a long-range effort, and do not support many conclusions at this time. Our analysis will require many more examples to be meaningful, and we bet that as we accumulate more data, this picture will change quite a bit before stabilizing. However, the data do indicate how we are gathering evidence for our claims, and some trends are already visible.

5.1 Effort

The data in this section are meant to support the claim that the effort to build an application with our tools is less than without them. Figure 5-1 lists -- for each application -- the time it took to create the initial application, integrate that application into the SBF framework (this includes the effort to create or refine mechanisms and/or their associated knowledge-acquisition tools), create the application with our framework, create a full-scale application, and the time it took a programmer to re-create the application.

Application	Develop Applica-tion (days)	Integrate into SBF (days)	Create using SBF (days)	Create full-scale (days)	Re-create by pro-grammer (days)
Qualifier	250	159	1	4	
SWSizer	--	3	1	9	
Candlestick	--	103	2	17	
PWSizer	--	0	1	9	
SWSelector	200	11	1	7	
Layout	69	63	1	1	
Retriever	57	19	1	1	
Scheduler	35	62	1	1	
Spark	65	239	1	1	
PC-Config	150	60	3	10	

Figure 5-1 Effort to Create Applications With and Without Our Framework

There is strong evidence that once we have put the necessary mechanisms into our framework, the effort to build an application is very small compared to the effort to build the application without the mechanisms. These data are confounded with the effort to understand the task and gather and organize appropriate information for building the application, and we therefore intend to have experienced programmers rebuild some of the applications using all the organized information to compare more directly with the effort numbers we have collected so far.

There were 4 applications that were built exclusively from reused mechanisms, and did not need a significant investment in building new mechanisms: SWSizer, PWSizer, SWSelector, and Retriever. Each of those applications used the same set (or a subset) of mechanisms as one of the others. These applications therefore were built with a very small effort: 10 - 20 days including analysis, extending mechanisms, and interacting with the knowledge acquisition tools. Our strong claim is that as we increase the number of mechanisms in the library, we will be able to build more and more applications with this little effort.

5.2 Application Scope

Figure 5-2 presents the application scope of our mechanisms. Mechanisms with a high application scope are those used in a large number of applications (for example, classify, select, and dialog-mgr). Those with very small application scope (the top 6 in the figure) should perhaps be generalized or combined with others to increase their application scope. We now need to gather this type of evidence for traditional programming languages and highly specialized tools.

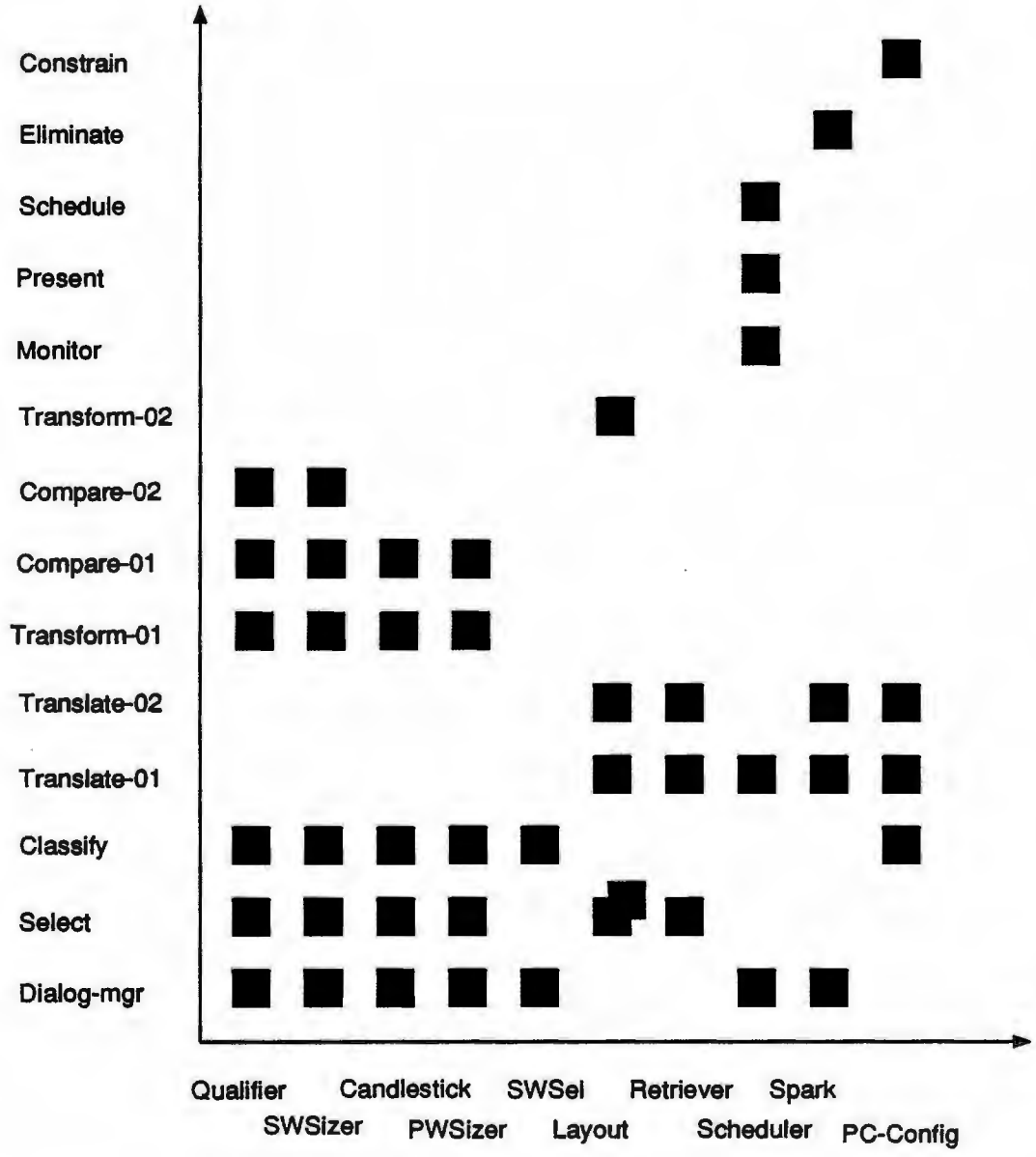


Figure 5-2 Application Scope

5.3 User Scope

Figure 5-3 shows the data indicating who has used our framework in creating the applications. The people in the SBF team are intimately involved in building the framework but are not task experts. A developer is a task expert but has no (or very little) knowledge of our framework. A developer may or may not have programming experience. Non-programmers represented in Figure 5-3 had less than one year of experience programming.

Application	Built by SBF team (success/attempt)	Built by Developer [programmer] (success/attempt)	Built by Developer [non-programmer] (success/attempt)
Qualifier	1 / 1		
SWSizer		1 / 1	
Candlestick	1 / 5	3 / 3	2 / 3
PWSizer		1 / 1	
SWSelector		1 / 1	
Layout	1 / 1		
Retriever	1 / 1		
Scheduler	1 / 1		
Spark	1 / 1		
PC-Config	1 / 1		
Success Ratio	7 / 11	6 / 6	2 / 3

Figure 5-3 Success of Different Types of Users in Building Applications

These data are preliminary but perhaps hint that it might be as easy for developers with little knowledge of our framework and the technical basis of our tools to create applications with our tools as it is for the SBF team. We would like to make the claim that knowledge of the task is more important than knowledge of our tools, but we need to collect more data. Other differences (such as greater motivation) can explain the apparent greater success of developers than of the SBF team in the Candlestick task. Or, it could be that one or more knowledge-acquisition tool (and mechanism) in Candlestick is hard for programmers to understand.

Figure 5-4 is derived from figure 5-3. It projects the results from figure 5-3 onto the individual mechanisms used by the applications.

Mechanism	Built by SBF team (success/attempt)	Built by Developer [programmer] (success/attempt)	Built by Developer [non-programmer] (success/attempt)
Dialog-mgr	4 / 8	6 / 6	2 / 3
Select	5 / 9	5 / 5	2 / 3
Classify	4 / 8	6 / 6	2 / 3
Translate-01	5 / 5		
Translate-02	4 / 4		
Transform-01	2 / 6	5 / 5	2 / 3
Compare-01	2 / 6	5 / 5	2 / 3
Compare-02	1 / 1	1 / 1	
Transform-02	1 / 1		
Monitor	1 / 1		
Present	1 / 1		
Schedule	1 / 1		
Eliminate	1 / 1		
Constrain	1 / 1		

Figure 5-4 Success of Different Users Analyzed for Each Mechanism

There is no evidence yet to indicate that some mechanisms (and their knowledge-acquisition tools) are harder or easier to use than others. The mechanisms with low success rates by the SBF team are those in the Candlestick application, the only application tested widely on the SBF team.

Several of the mechanisms (and their knowledge-acquisition tools) have been used by non-programmers to successfully build application programs. Therefore, our user scope is somewhat larger than that of the programming constructs used by traditional programming languages. We now need to collect similar data for other approaches to show, for example, how our approach compares to that of specialized inference engines.

5.4 Functional Scope

We stated in the introduction that one of the problems with small, general programming constructs is that a lot are needed to cover an application. Thus, the task of determining how to combine the reusable parts into a working program requires significant programming skill. Figure 5-5 lists for each of our mechanisms the average number of mechanisms it is configured with, the smallest and largest number of mechanisms it is configured with (range), and the number of applications using that mechanism.

Mechanism	Ave. size of configuration	Range	Number of examples
Constrain	4	--	1
Eliminate	4	--	1
Schedule	5	--	1
Present	5	--	1
Monitor	5	--	1
Transform_02	5	--	1
Compare_02	6	6-6	2
Compare_01	5.5	5-6	4
Transform_01	5.5	5-6	4
Translate_02	4	3-5	4
Translate_01	4.2	3-5	5
Classify	4.5	2-6	6
Select	4.8	3-6	6
Dialog-mgr	4.6	2-6	7

Figure 5-5 Size of Mechanism Configurations Analyzed for Each Mechanism

We do not have evidence showing significant differences between mechanisms, and, in the absence of data from use of other types of programming constructs, it is difficult to say what the functional scope of our mechanisms means. It is, however, between the extremes of a single, large, monolithic inference engine and the tiny program statements of traditional programming languages like FORTRAN, OPS5, Lisp, Prolog.

5.5 Usefulness

As we stated in the beginning of this paper, effective programming is the creation of programs that are useful to the end-users. It is our goal with all applications we create to identify end-users and evaluate how useful are the programs we build. All are intended to help workers be more productive in their

work. To date, however, we do not have much data showing the usefulness of our applications, as we are in the process of designing field tests for them. We have presented three applications to end-users thus far (Candlestick, SWSelector, and Spark) and all have been used somewhat, but we have not yet established procedures to ensure that our applications can be evaluated systematically. Devising a workable end-user test and evaluation plan is one area where we will be putting a strong effort over the next year.

6 Conclusion

There are other research efforts with approaches similar to ours. The generic task approach of (Chandrasekaran, 1983; Chandrasekaran, 1988) and colleagues builds applications from generic components (mechanisms, in our terminology). This work emphasizes a few generic tasks, large in functional scope, and applicable to a large range of tasks. Our approach concentrates on a greater number of less general mechanisms, in the hope of being able to provide more assistance with specializing the mechanisms, thus requiring less skill of the user.

The KADS-II project (Breuker, Wielinga, van Someren, de Hoog, Schreiber, de Graf, Bredweg, Wielemaker, Billault, 1989; Wielinga, Schreiber, Breuker, 1991) is developing a library of building blocks (knowledge sources in their terminology) that appear to be of similar scope as our mechanisms, and they have built a somewhat formal framework of interrelated models to guide the process of building applications. The biggest difference is that the KADS-II framework is aimed to help a "knowledge engineer" through a well-specified process to design the application. Our research aims to provide more proactive assistance in application design, enabling task experts with less programming expertise to build their own applications. Other work attempting to address both a large scope of users and a large range of applications simultaneously but with different approaches include (Malone, 1988; Musen, 1991; Yost, 1989).

We have built applications with reusable mechanisms that are usable by non-programmers to implement applications in less than a month. We did this by creating mechanisms that are intermediate in functional size between the programming constructs of programming languages and large inference engines. We have set up an experimental method to evaluate our approach, and our preliminary data do not contradict our claims, and lend a little support, particularly in the claim that it takes less effort to build an application within our framework than without it.

However, the data presented in the previous sections are based on a relatively small number of applications (10) and mechanisms (14). Although those experiments represent a large effort (greater than 6 person years; see table 5-1) a lot more data points are needed for a sound statistical analysis. We are just at the beginning of developing a theory of mechanism. Each new application leads to changes in our framework, or -- as a mechanism gets reused by another application -- makes it necessary to refine the existing mechanisms. We will continue using Spark, Burn, and FireFighter to create application systems and, thus, refining the definition of mechanisms. We believe that this approach will lead us to a definition of usable and reusable mechanisms.

One of the biggest problems we have in our approach is getting people to make contact with our activity models in Spark. Our latest efforts have shown that it is difficult for real people to understand somewhat generic, abstract models of their task unless they have participated in the creation of that model. Real people doing real tasks are buried in the detail of that task; details that make their task unique. These details are exactly those by which a person would recognize their task and yet these are the same details which an abstract model has removed. We are now moving toward use of dynamic models created by the developers in defining their task within our modeling framework. Our Spark task then becomes mapping the features of this new model to those attributes of activities we already know how to automate. An initial description of our refined understanding of the Easy Programming Framework can be found in (Dallemagne, Klinker, Marques, McDermott, Tung, 1991).

Acknowledgements

This paper would not have been possible without the help of our co-workers in this project: Glee Cameron, Charlie Reed, and Tina Whitney. All made significant contributions to the work described here.

References

Bennett, J. (1985) "ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System," *Journal of Automated Reasoning*, Vol. 1.

Birmingham, (1988) "Automated Knowledge Acquisition for a Computer Hardware Synthesis System," *Proc. 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada.

Boose, (1984) "Personal Construct Theory and the Transfer of Human Expertise," *Proc. 4th AAAI*, Austin, Texas: Morgan Kaufmann.

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Graf, P., Bredweg, B., Wielemaker, J., and Billault, J.P. (1989) "Model-Driven Knowledge Acquisition: Interpretation Models", ESPRIT Project 1098, Deliverable task A1, Memo 87, VF Project Knowledge Acquisition in Formal Domains, Amsterdam.

Chandrasekaran, B. (1983) "Towards a Taxonomy of Problem Solving Types," *AI Magazine*, Vol. 4, No. 1.

Chandrasekaran, B. (1988) "Generic tasks as building blocks for knowledge-based systems: The diagnosis and routine design examples," *The Knowledge Engineering Review*, Vol. 3, No. 3, pp. 183-210.

Clancey, W.J. (1983) "The Epistemology of a Rule-Based Expert System -- a Framework for Explanation," *Artificial Intelligence*, Vol. 20, No. 2.

Dallemagne, G., Klinker, G., Marques, D., McDermott, J., Tung, D. (1988) "Making Application Programming More Worthwhile," to appear in *Knowledge Engineering and Cognition*, Schmalhofer, F., Strube, G., Wetter, Th., eds., Springer Verlag.

Davis, R. (1979) "Interactive Transfer of Expertise: Acquisition of New Inference Rules," *Artificial Intelligence*, Vol. 12, No. 2.

Eshelman, L. (1988) "MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems," in *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, ed., Boston, Dordrecht, London: Kluwer Academic Publishers.

Freeman, P. (1987) "A Conceptual Analysis of the Draco Approach to Constructing Software Systems," *IEEE Transactions of Software Engineering*, Vol. 13.

Klinker, G. (1988) "KNACK: Sample-Driven Knowledge Acquisition for Reporting Systems," in *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, ed., Boston, Dordrecht, London: Kluwer Academic Publishers.

Klinker, G, Bhola, C., Dallemagne, G., Marques, D., and McDermott, J. (1991) "Usable and Reusable Programming Constructs," *Knowledge-Acquisition*, Vol. 3, No 2, Pages 117-135.

Krueger, C. (1989) "Models of Reuse in Software Engineering," *Technical Report CMS-CS-89-188*, Carnegie Mellon University.

Malone, T. (1988) "Object Lens: a 'Spreadsheet' for Cooperative Work," *ACM Transactions of Office Information Systems*, Vol. 6, No. 4.

Marcus, S. (1988) "SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems," in *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, ed., Boston, Dordrecht, London: Kluwer Academic Publishers.

Marques, D, Dallemagne, G., Klinker, G., McDermott, J., and Tung, D. (1990) "Easy Programming: Empowering People to Build Their Own Applications," *to appear in IEEE Expert.*

Musen, M., Tu, S. (1991) "A Model of Skeletal-Plan Refinement to Generate Task-Specific Knowledge-Acquisition Tools," *Machine Learning*, Vol. 4, No. 3/4.

Musen, M. (1989) "Automated Support for Building and Extending Expert Models," *Technical Report KSL-91-05*, Knowledge Systems Laboratory, Stanford University.

Rich, C., and Waters, R. (1990). *The Programmer's Apprentice.* Addison-Wesley.

Wielinga, B.J., Schreiber, G., and Breuker, J.A., (1991) "KADS: A Modeling Approach to Knowledge Engineering," ESPRIT Project P5248 KADS-II/T1.1/PP/UvA/008/1.0, University of Amsterdam.

Yost, G. (1989) "A Problem-Space Approach to Expert-System Specification," *Proc. 11th IJCAI*, Detroit, Michigan.

Towards an Analysis and Classification of Approaches to Knowledge Acquisition from Examination of Textual Metaphor

MARY A. MEYER* AND RAY C. PATON+[1]
*MS F600, Statistics Group, Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA
email: myym@lanl.gov

+Dept. Computer Science, The University of Liverpool, Liverpool L69 3BX, U.K.
email: rcp%and.cs.liv.ac.uk

Abstract

Metaphors are appropriate to understanding approaches to knowledge acquisition because knowledge engineers' choice of metaphor reflects and affects their thoughts, and through these, their practices in the field. Salmond, an anthropologist, provides a methodological model of how metaphor in text can be manually analyzed and used to classify the text according to its analytical perspective on the phenomena of study. We expand upon Salmond's method of metaphorical analysis and perform detailed analyses of text on knowledge acquisition and ethnomethodology. We develop Salmond's ideas for classifying perspectives and propose a three-part classification scheme: task-, concept-, and domain-based.

1. Introduction

The abstract aspects of science are frequently described using figurative language, especially metaphors. Metaphors are used by people to convey information beyond the literal transcription of their words. Scientists use both models and metaphors to describe one thing in terms of another and both refer to entities, whether real, hypothetical or totally fictional, in the world. Metaphor itself is a form of language use and is understood in terms of a complete utterance rather than at the level of an individual word.

The choice of metaphor reflects and affects people's thoughts; thus, metaphor enters into the realm of practice and actions. Metaphors can be used to articulate different aspects of reality, from different angles and in different ways. For example, organization theorists (e.g., Koch & Deetz, 1981; Smircich, 1983) demonstrated how certain pervasive metaphors, such as *organization-as-machine* or *organization-as-organism*, have greatly affected the way problems are described and solved by people in organizations. They show how a shift to another metaphor, such as *organization-as-culture* not only affects what is investigated, but also the theory and ontological commitments that are formed. The analysis of metaphorical language within a newly emerging science allows an investigator to assess the current status of theory and practice and anticipate possible future directions within the field of knowledge (Paton, Nwana, Shave, Bench-Capon & Hughes, 1991).

In light of the previous discussion, we propose that the metaphor in knowledge acquisition text be analyzed. Metaphorical analyses can provide us with insight into the assumptions that we, as practitioners of knowledge acquisition, make about the properties and nature of knowledge. Metaphorical analyses can also be used, as Salmond has, to point out the cultural base or bias of many of these assumptions. In addition, metaphorical analyses can been applied to various

[1] This paper was jointly authored, and the authors' names appear in alphabetical order.

knowledge acquisition approaches to classify them according to their perspective. By knowledge acquisition approach, we mean any technique, method, or field of study that is applied to the field of knowledge acquisition. For instance, we illustrate the classification with the ethnomethodological approach because it has recently been applied to knowledge acquisition. We hope that classifying an approach will enable the individual to understand it in relation to other approaches, note its assumptions, and anticipate its strengths and problems.

This paper is organized as follows. In section 2, we present background information on the origins of metaphor and its use in scientific thought. In addition, we describe Salmond's method of semantic analysis. In section 3, we extend Salmond's work and propose a classification scheme for approaches to knowledge acquisition—task-, concept-, and domain-based. The foci, advantages, and problems of each of these three approaches are outlined. In section 4, we describe and illustrate metaphoric analysis using informal text on knowledge acquisition. We perform a sample analysis and classification of the ethnomethodology approach. In section 5, we discuss the uses and implications of metaphoric analyses with regard to knowledge acquisition. Section 6 is a brief summary.

2. Background

2.1. HISTORICAL ORIGINS OF METAPHORS RELATING TO KNOWLEDGE

Metaphor has been linked to the functioning of the human mind since the ancient Greeks (Snell, 1953, p. v). They used metaphorical terms to name the life of the mind, as follows:

> At first the mind is understood by analogy with the physical organs and their functions: the *psyche* is the breath . . . the *thymos* is the organ of internal (e)motions, and the *nous* is the mind in its capacity as an absorber of images. Knowledge (*eidenai*) is the state of having seen; recognition (*gignoskein*) is associated with sight, understanding (*synienai*) with hearing, know-how (*epistasthai*) with practical ability. (Snell, 1953, p. 198)

Once these distinctions were made, they were readily passed down into modern European languages in the forms we know today. Thus, we think of knowledge and cognition in terms of *seeing* and tend to use language associated with this sense. In sum, the history of consciousness has been bound up with metaphor.

2.2. IMPORTANCE OF METAPHOR TO SCIENTIFIC THOUGHT

> Metaphor and simile are the characteristic tropes of scientific thought, not formal validity of argument . . . (Harré, 1986, p. 7)

The purpose of this section is to briefly discuss the importance of metaphor in scientific thought.[2] There is a vast bibliography about metaphor, language and thought (e.g., Johnson, 1981) and a variety of theories have been proposed in an attempt to explain its nature. Following Soskice (1985, p. 15), we take metaphor to be "that figure of speech (trope) whereby we speak of one thing in terms which are seen to be suggestive (by a competent speaker of the language) of another". Metaphor is a form of language use. It is neither a mental event nor a physical object, but it provides the context for making comparisons and describing differences between objects (i.e., referents in the real world—both observable and non-observable). It is through metaphor that analogies and similes are made. The idea that metaphor is an ornamental property of language that is replaceable (without loss of meaning) by literal transcription has been thoroughly discredited (Harré, 1986). Scientific metaphors and models have a lot in common with each other (Harré, 1970; Aronson, 1991; Way, 1991). However, metaphors can only be language based, models need not be.

---

[2] We are informed by a reviewer that some of these ideas date back to the 1920's and a book by Max Scheler entitled *Problems of a Sociology of Knowledge.*

Metaphor serves several functions in scientific language. First, metaphor can be catachrestic. Human experience is bigger than the human literal vocabulary which is used to describe, discover and communicate it (Ortony, 1975). As such, we use metaphors to fill this vocabulary gap, for example, to introduce theoretical terminology where none previously existed (Boyd, 1979). However, metaphor can be much more than catechrestic. Ricoeur (1973) argued that it is able to redescribe reality through the tension existing between sameness and difference; the old remains but is seen in a new light. This ontological role of metaphor in science has been stressed by several philosophers (such as, Aronson, 1984; 1991; Harré, 1986; 1990; Rothbart, 1984; 1991). For example, the computational metaphor in the biosciences and psychology pervades much of the current language about the brain; Arbib's description of *The Metaphorical Brain* (Arbib, 1972; 1989) is but one example of the impact of cybernetic thinking on the brain sciences. Thinking of the brain as a computer not only influences how we study it but also how we formulate models and describe research problems (see Genter & Grudin, 1985). A third function of metaphor is that it can be didactic. Teachers intentionally manufacture metaphors in order to promote understanding by their students (Lewis, 1939). In this case, much of what we assume has been learned *literally* is actually learned *metaphorically*. The impact of scientific metaphors on the educational process can be both beneficial and disastrous (Paton, 1988).

Metaphorical analysis provides an invaluable technique for organizing the language of a domain. Consider a simple example. Morgan (1980) analyzed the language of the machine metaphor in organization theory as found in the writings of classical management theorists and in Weber's specification of bureaucracy as an ideal type. He extracted the following principles as implicit in the author's use of the machine metaphor:

1) Machines are rationally devised for performing work in pursuit of pre-specified ends.
2) Goals and means-end relationship as a purposive rationality.
3) Model details are drawn from mechanical concepts.
4) Operation of the whole bureaucratic enterprise is judged in terms of efficiency.
5) The organization is viewed as a somewhat closed, static structure.

Alternative metaphors, such as *loosely coupled systems, ecological systems, theaters, cultures, language games,* or *texts* provide ways of looking at organizations which elude the machine metaphor. Morgan notes how a functionalist way of theorizing is based on the assumption that the reality of organizational life rests in a network of ontologically real relationships. The idea of *organization-as-culture* or *organization-as-theater* when occurring within a functionalist framework, has non-concrete entities. This is more clear when we think of *organizations-as-language games*, in which case the objects in the ontology are words, thoughts and actions.

This type of analysis can provide insights into the way models are constructed. For example, the language of the machine metaphor can be detected across many scientific domains, such as biosciences (Paton et al, 1991a) and economics (Paton, Nwana, Shave & Bench-Capon, 1991b). One way of assessing the nature of models in a particular domain is to investigate language associated with a particular metaphor. Such an investigation could yield, for example, the following attributes for the metaphors of *machine, text,* and *organism.*

Machine	goal, design, purpose, mechanism, equilibrium, control, efficiency, optima.
Text	code, interpretation, language, grammar, translation, meaning, context.
Organism	growth, order, organization, adaptability, complexity, openness, individuality.

It is important to note that complex domains will tend to be polytypic in nature; that is, they will use language associated with more than one metaphor. For instance, the language of artificial neural networks includes that of *machine, text,* and *organism.* Metaphors of machine, text, and organism are classified as belonging to the *systemic* type of metaphor (Paton, 1991) in terms of such concepts as interrelationship, organization and collective behavior. When authors refer to the organization of artificial neural networks, they use systemic metaphors; when they mention behavior, they use spatial language (e.g., *local minima, gradient descent, changes to landscape*). Thus, this domain is polytypic and uses different systemic and spatial metaphors.

Metaphorical language is highly pervasive in our language use. So-called *dead metaphors*, that is, language which originated metaphorically but has gained a (so-called) literal usage, are not so *dead* as they might seem (see Soskice, 1985). One analysis of metaphorical language is that of Lakoff and Johnson (1980). For example, in their investigation of the language used in association with the idea of an argument, they note it is described:

as a building	We needed to construct a strong argument . . . only to find that it collapsed.
as a journey	We have set out to prove that . . .We have arrived at a disturbing conclusion.
as war	The claims were indefensible . . . every weak point in the argument was attacked.
as a container	This argument did not hold water . . . it was full of holes.

Analysis not only shows how certain ideas persist, it also indicates that these ideas have certain properties or entailments associated with them. These convey an emphasis although it must be noted that a given utterance is likely to use a mixture of types. For example, *argument-as-journey* gives ideas about progress whereas *argument-as-container* gives ideas about content. One important implication for a classification of approaches to knowledge acquisition is that certain figures of speech will provide pointers to underlying assumptions made and the models that result.

2.3. SALMOND'S METHOD OF SEMANTIC ANALYSIS

Salmond, a semantic anthropologist,[3] applied Lakoff and Johnson's approach to an analysis of the nature of knowledge in anthropology. Salmond's anthropological approach to the analysis of metaphor in text is not the only anthropological method to be used in the field of knowledge acquisition. Recently anthropological approaches have been proposed for use in knowledge acquisition (Paton, Nwana, Shave, & Bench-Capon, 1990b). For instance, Forsythe and Buchanan (1989, p. 436) reason that anthropological techniques are applicable because of the overlap between the goals of artificial intelligence and anthropology. That is, both fields seek to investigate "how human beings understand, organize, process and make use of symbolic information." Additionally, both fields rely on the same basic data gathering methods—interviews with individuals, text analysis, and observations. They note that anthropology, because of its longer history and extensive literature on its methods, has much to offer the field of knowledge acquisition.

Central to Salmond's paper (1982) are the following four points:
that language is an instrument for the negotiation of meaning;
that this argument applies also to language used in scientific discourse;
that figurative language, particularly metaphor, plays an important role in scientific communication;
and that our metaphors for knowledge are culturally specific, thus constraining our knowledge of knowledge.

2.3.1. Salmond's procedures in conducting a semantic analysis of metaphor

Salmond randomly selected theoretical texts in anthropology and performed the following procedures:

- identification of the trope in the text. For example, Salmond begins with the metaphor, or claim, that *knowledge is a landscape.*
- identification of what the trope entails in the culture. For instance, the claim that *knowledge is a landscape* (1982, p. 67) entails that knowledge is a territory, which entails that it has spatial existence, which further entails in common-sense Western culture that it is three dimensional and static (i.e., the land does not move or change)[4].

---

[3] Semantic anthropologists have been concerned with the "identification of semantic domains represented by the lexical items (vocabulary) of the languages they have studied " (Hunter and Whitten, 1976, p. 345). Semantic anthropology is more frequently termed cognitive anthropology in the United States.

[4] Instead, wind and water are viewed as dynamic in common-sense Western culture; hence the expression *the winds of change.*

- identification of related tropes, such as those which would be predicted by the above entailments. These are some of the related tropes that she finds: *knowledge is territory, intellectual activity is a journey,* and *understanding is seeing.*
- determining how the tropes are linked or finding their common themes. Salmond describes the above tropes as being linked in the following way. If knowledge is a territory, it follows that the traveller will travel in acquiring it, and that in traveling, the traveller will look about and notice features of the domain. The traveller will have a perspective, a close or distant way of seeing. This further implies that the viewer is separate from the viewed and that the "physical existence of the domain is not in doubt" (1982, p. 73). Through this process, she pinpoints the theme that these metaphors share—that knowledge is a physical activity in a physical environment. She is able to eventually propose that theoretical orientations in anthropological discourse involve "metaphors about location in a physical landscape, where space-time properties are those of common-sense Western perceptions" (1982, p. 6). She concludes that the anthropologists depict and understand knowledge *as a theoretical landscape.*
- comparing tropes on common concepts, such as knowledge, across cultures. For instance, Salmond compares the anthropological, and largely Western, view of knowledge to that of the Maori, a Polynesian population in New Zealand. In contrast to anthropologists, the Maori view is of *knowledge as wealth;* knowledge is locked into objects and landscape with a power of naming. "The 'landscape' account presents the anthropologist as a detached intelligence, working to domesticate and master an objectified world, whereas in Maori thinking, truth is always contingent upon situational factors, knowledge claims are tied to power, the environment has a right of reply and one is vulnerable to its rebuttals" (1982, p. 85).

2.3.2. Proposal that textual descriptions of knowledge acquisition approaches can be analyzed for metaphor and classified

In line with Salmond's work, we propose that textual descriptions of knowledge acquisition and approaches to it can be analyzed to determine how they view knowledge and the corresponding knowledge engineering activity. We further propose that these approaches to knowledge acquisition can be classified by their perspectives, as portrayed through their use of metaphor.

Salmond noted that there were very few metaphors in all the anthropology texts that did not relate to the dominant metaphor of landscape. If we can assume then that authors within a field are consistent in the way that they use metaphor, we may validly select a small sample of text for our analyses.

3. Description of Classification of Approaches to Knowledge Acquisition

3.1. INTRODUCTION OF SALMOND'S THREE APPROACHES

Salmond semantic analysis of metaphor in anthropological text led her to propose that different theoretical orientations are expressed as "different vantage points in epistemological space" (1982, p. 74). Using examples from anthropology, she proposed three types of orientations:

- the structural approach, where the people are viewed as from a great height. The Structuralism of Levi-Strauss would be an example of this approach. Levi-Strauss was interested in reducing the diversity of behavior in art and ritual through structural analyses to underlying rules or oppositions and using these structural principles to analyze art and ritual across cultures. Thus, a structural approach, per se, would not focus at the level of the individual but at a more abstract one, where as Salmond notes, the natives would appear like ants (see Fig. 1). Salmond adds that each approach's perspective implies some status relationship between the researcher and those who are observed; in the structural approach, for instance, the researcher is "hierarchically superior" (1982, p. 74).
- the ethnoscientific approach, where the researcher tries *to get into the head* of the native and *see the world from his/her eyes.* This approach is the traditionally used by ethnographers in the field. Frequently, ethnographies use the natives' terms for things and attempt to portray the

meaning that the natives attribute to events. As Salmond notes, the anthropologists try to replicate the natives' *world view* by adopting their vantage point. Participant observation, the traditional field method, is one of the means by which anthropologists *put themselves in the natives' place*. They believe that observing and participating in the natives' everyday lives will allow them to have an insiders' view (Meyer, Picard & Ross, 1990, p. 23-2). In the ethnoscientific approach, the researcher seeks to efface him or herself to more fully learn the insider's view.

- the hermeneutic approach, where Salmond proposes that the orientations of the researcher and those observed would be merged. While she gives no examples of this approach, the idea is for the different perspectives of the researcher and the observed to be overlapped. In this approach, Salmond proposes that the researcher and the observed would be of equal status.

3.2. EXTENSION OF SALMOND'S APPROACHES TO A CLASSIFICATION SCHEME OF TASK-, CONCEPT-, AND DOMAIN-BASED APPROACHES

The theory and practice of knowledge acquisition can be examined according to the emphases placed on this activity as revealed by the language practitioners use. Salmond's proposal of the three approaches was used as a starting point for developing a classification of approaches to knowledge acquisition (Paton et al, 1990b). The task-, concept-, and domain-based approaches described below roughly correspond, respectively, to Salmond's structural, ethnoscientific, and hermeneutic approaches (as shown in Fig. 1).

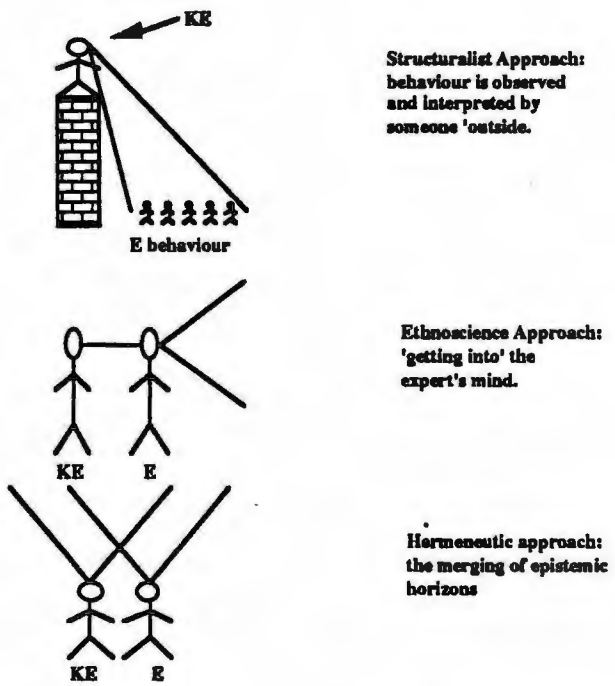


Fig. 1. Salmond's three anthropological orientations applied to knowledge acquisition.
(This figure was excerpted from Paton et al, 1990b, p. 2.)

3.2.1. Task-based approach

This approach focuses on the behaviors (i.e, that which is observed) that are needed to bring about solutions to problems. These behaviors relate to what Newell (1982) describes as the Knowledge Level; that is, what is done, not how it is done. Solutions to problems are brought about by constructing models, not of what is *in an expert's head*, but rather of how the behaviors are properly combined together. We may summarize certain key features as follows:

- The overall emphasis of this approach is construction of a computable problem solving model.

- The key cognitive objects are tasks.
- The central focus is computable (Knowledge Level) models which solve problems.
- In many ways the role of a knowledge engineer should be passive, s/he is the facilitator/translator of expert behavior to computable code.

There are problems with this kind of approach. First, there is more to a domain than its tasks. Emphasis on tasks alone will detach the meaning of utterances about the behavior of a system from the cognitive and social environment in which the system is purported to operate (Gaines, 1989). Put another way, how is the knowledge in the system related to the real world domain? This issue is partly addressed by the move towards the incorporation of qualitative modeling into second generation expert systems (Clancey, 1989) but there is a problem concerned with what a model is modeling and how the former is related to the real world domain. Second, the notion of level or layer requires more serious philosophical investigation. The idea of knowledge level has its origins within the computational metaphor and yet talks in terms of a spatial metaphor. Knowledge that embeds a task within a coherent problem solving model requires metaknowledge beyond the knowledge level.

The task based approach is typically structural. It emphasizes observable behaviors and defines models on them according to prescribed rules of combination.

3.2.2. Concept-based approach

This approach focuses on what is *in an expert's head.* Metaphors such as mining and digging have been used in an attempt identify how knowledge is acquired. The models constructed in this approach are cognitive models of human cognitive structures. This contrasts with the former approach where the cognitive models need not reflect the cognitive processes of human problem solvers, only their observable (codeable) behaviors.

- The overall emphasis of this approach is clarification of a cognitive problem solving model.
- The key cognitive objects are *concepts in the head.*
- The central focus is on experts who solve problems.
- In many ways the role of a knowledge engineer should be passive, s/he is the facilitator/translator of expert conceptual structures to (possible) computable code.

Problems associated with this kind of approach concern the idiosyncratic nature of human cognition. The focus on *getting something out of someone's head* is dependent on the *extraction* or *mining* technique employed. As a result, the quality of a computational artifact may never be any better than the quality or applicability of the tool/technique that has been used. A further problem is how to make sense of the extracted data. In the latter case, we build models. However, these models need more than what we observe because we cannot directly observe the thinking process. We have to make assumptions about such models, as in the case of parallel distributed processing in relation to microcognition.

The concept-based approach is typically ethnoscientific. It emphasizes relations between terms and defines models on them at the lexical (rather than whole utterance) level of description.

There are several examples of each of these approaches to knowledge acquisition in the literature. A third approach, which is less well documented, though relevant to knowledge acquisition, is the domain-based or hermeneutic approach.

3.2.3.. Domain-based approach

This approach focuses on the domain itself and seeks to merge the epistemic perspectives of the expert and the knowledge engineer. Clearly the former is a lot more knowledgeable than the latter. The models constructed in this approach are of the domain and specifically, the expert's understanding of the domain.

- The overall emphasis of this approach is characterization of the domain.
- The key cognitive objects are concepts which have a referential basis in the real world.

- The central focus is on domains which bring about the solutions to problems.
- In many ways the role of a knowledge engineer should be active, s/he is the interpreter of someone else's epistemic horizon on *what is out there.*

This kind of approach has also been described as hermeneutic in that it emphasizes interpretation through the sharing of a *similar epistemic horizon* (Paton, Nwana, Shave & Bench-Capon, 1991c). For the purposes of knowledge acquisition, the focus of attention is the domain. The domain includes the real world referent as well as the cognitive constructions of the expert and knowledge engineer. The basis of the hermeneutic approach lies in the proposal that a domain shares common characteristics with a text and a society. Ricoeur (1971, 1981) proposed and developed a hermeneutical approach to social investigation. He argued that if there were specific problems raised by the interpretation of texts because they are texts and not spoken language, and, if these problems were the ones which constitute hermeneutics, then the human sciences may be said to be hermeneutical in that:
1) They consist of objects which display some of the features constitutive of text as text, and,
2) Their methodology develops the same kind of procedures as those of text interpretation.
For instance, *society-as-text* is a useful metaphor for providing a fresh way to look at questions of explanation, understanding and interpretation. Ethnographers become cultural translators of the societal ways. In the same way, *domain-as-text* becomes a useful metaphor for similar purposes.

Advantages with this kind of approach are:
- It requires and seeks to provide the means for communication between the expert(s), the knowledge engineers and other parties (such as clients). At a general level it provides a means for arriving at the basis for a common ontology of the domain.
- The approach has breadth: without sufficient breadth to begin with the knowledge engineer may miss part of the domain altogether.
- Knowledge engineers are not passive, dumb, non-interactive agents. Rather, they bring certain assumptions and preconceptions with them when embarking on knowledge acquisition. We argue that it is crucially important to make this background information explicit. This is achieved by requiring that initial approaches to the characterization be broadly based.
- The broad approach relates to the overall characterization of the domain. On interacting with a domain expert, the knowledge engineer may have to approach elicitation (in part) as meets the needs, idiosyncrasies and requirements of the expert or his/her schedule.
- An appreciation of the real world complexity and the cognitive objects used to refer to it, can help a knowledge engineer place the domain in context (both cognitive and referential).

We argue that the three approaches to knowledge acquisition are all relevant to the knowledge engineering enterprise. Each has a different emphasis which is necessary to complete a knowledge based system development from that which exists in the real world, through the cognitive models people construct, to the subsequent system.

4. Sample Analysis and Classification of Knowledge Acquisition Text Using Metaphors

4.1. SAMPLE ANALYSIS OF METAPHOR IN KNOWLEDGE ACQUISITION USING RECENT KA NEWSLETTERS

We chose to do a metaphorical analysis of general text on knowledge acquisition, for two reasons: first, this analysis would be at the approximate granularity as Salmond's semantic analysis of anthropology text, thus allowing us to compare the results of these analyses (as shown in 4.1.3); and second, it would allow us to learn the meaning of some utterances, such as knowledge, tool, theory, as denoted by their terms. We planned to examine the use and denotation of these terms within texts on particular approaches to knowledge acquisition, such as ethnomethodology (section 4.2.3).

We chose the two most recent editions of the KA Newsletter (vol. 2, no. 2 and vol. 3, no. 1) because they include the summary of many papers/meetings, use the most current and informal language (such as in the working group slogans), and are, very importantly, short.

4.1.1 Application of semantic analysis of metaphor to knowledge acquisition text
We will describe our steps in doing this semantic analysis of metaphor to allow the reader to follow and evaluate our procedures and conclusions.

Our first step was to identify some tropes pertaining to tools, techniques, theory, knowledge, knowledge acquisition, expert, and knowledge engineer. The following metaphors were found, mainly in the slogans from the working groups in Newsletter vol. 3, no. 1:

> Theories without tools are empty, tools without theories are blind.
> Schools for tools.
> Posthoc theorization is rationalization, prehoc theorization is doing it right.
> Raiders of the lost techniques.
> ML is interested in the theory of computation for the algorithms. KA is interested in the cognitive theory for the model.
> He [Gaines] sees the interaction of machine learning and knowledge acquisition as a tight web, where machine learning helps in the transformation of the conceptual nets acquired by the KA techniques into formal representations." (vol. 2, no. 2, p. 9)

The second step was to determine what these tropes entailed, not only in Western culture but within the subculture of knowledge acquisition. As an aid to thinking about what each referent entailed in Western culture, these terms were looked up in Webster's New World Dictionary.

> tool: an implement, instrument, or utensil held in the hand and used for cutting, hitting, digging, rubbing, etc. . . . anything that serves in the manner of a tool.: a means: as, books are a scholar's tools.
> technique: method of procedure in . . . carrying out a scientific or mechanical operation.
> theory: a mental viewing; contemplation. . . . a systematic statement of the principles involved. . . . a formulation of apparent relationships or underlying principles of certain observed phenomena which has been verified to some degree.
> knowledge: the act, fact, or state of knowing; specifically, acquaintance with fact, place. All that has been perceived or grasped by the mind; learning; enlightenment.

The above information stimulated us to think of additional entailments or properties for the above-mentioned terms. We then searched the Newsletters for tropes illustrating the entailments that we had generated. For instance, one common (Western) entailment of tools is that they are associated with doing different tasks. That is, tools are specialized (e.g., *tools of the trade*). The Newsletter illustrates the existence of this entailment for knowledge acquisition tools: ". . . development of tools that were specialized for the acquisition, representation, and interpretation of knowledge for one problem-solving method" (vol. 2, no. 2, p. 8). Additionally, one working group states that "no one tool/technique does it all"(vol. 3, no. 1, p. 11). As an interesting aside, the latter is given as the reason for knowledge engineers searching for usable tools, techniques, and theories from the social-behavioral sciences (p. 11). This theme of the knowledge engineers' search for tools is further found in the slogan "raiders of the lost techniques" (and, we propose, in the 1988 slogan, "fools for tools" and in the general use of the term *toolbox*).

Another property of a tool comes from the reason a person would use it—it extends a person's capabilities in performing some operation. From the quote in the above paragraph, we learn that tools somehow help in the acquiring, representing, and interpreting of knowledge. (We will return to this link between tools and knowledge in step 3.)

Another entailment of tools in general is that they can be manually or machine driven. The Newsletter mentions both possibilities, manual and automated, but usually only in connection with techniques (e.g., vol. 3, p. 15). The subtleties of this entailment are discussed below in the third step, the linking of tools to techniques.

In our search for tropes that would illustrate common entailments of tools, we stumbled across some tropes whose entailments were uncommon, at least in terms of the usual Western thinking. One such entailment of tools/techniques is that of *hardness* and *softness*. For instance, *hard*-coded tools are described as *brittle*. This unusual property of tool will be discussed further below in step three, the link between tools or techniques and knowledge.

Lastly, the Newsletter mentions as a desirable property, that the tools be used in combination to benefit one another. For instance, one working group states that they "do not want a toolbox where each tool is separated and cannot benefit from the other tools" (vol. 3, no. 1, p. 8).

In our third step, we searched the Newsletters for clarification of the relationship between (1) tools or techniques, (2) tools/techniques, and theory, (3) tools/techniques and knowledge or knowledge acquisition.

1) With respect to the link between tools and techniques, frequently tools and techniques are referred to in parallel, as equal but related terms (e.g., *Proceedings of the AAAI-90 Workshop on Knowledge Acquisition: Practical Tools and Techniques*). Sometimes the distinction is between *automated and manual techniques* (vol.3, no., p. 15) and other times between knowledge acquisition techniques (plus methods) and systems (e.g., *transfer/modeling of expertise-systems that obtain and model knowledge from experts. vs. transfer/modeling of expertise-manual knowledge acquisition methods and techniques* [p. 16]). Thus *tool* seems to be equated with system, or automation, while *technique* seems to refer to manual methods, such as interviewing, unless automation is specified; in which case, only the manual techniques would include such methods as non-computer aided interviewing. Because tools and techniques are generally referred to in parallel as a means of doing knowledge acquisition, we will lump tools and techniques together as one concept in examining their relationship to theory and knowledge, as described below.

2) The phrase about the practice of "theory fitting to tools" (vol. 3, no. 1, p. 11) allows us to interpret the first slogan quote which also links theory and tools—"Theories without tools are empty, tools without theories are blind." Thus, in the field of knowledge acquisition, theories are considered desirable or necessary components of tools. Because theory had been defined in the dictionary as a mental viewing or contemplation, tools without theory do not provide sight, or understanding. From the third slogan listed above, we learn that the theory in a tool is the reason or justification for selecting a particular tool; that if theory is fit to the tool after the tool has already been selected for an application, this practice is rationalization and, by implication, wrong.

3) To describe the link between tools/techniques and knowledge, we must backtrack to the properties of knowledge as these are described in the Newsletters. The ubiquitous term *knowledge acquisition* indicates that knowledge, an intangible, is viewed as something that can be acquired, or gathered in some fashion. Some of the other metaphors mentioned for knowledge acquisition, such as *chicken soup, filtering process, bottleneck*, at the Japanese meetings (vol. 3, no., p. 6) support this view of knowledge. Some other quotes, working group slogans (vol. 3), referring to knowledge are as follows:

> Machine learning and knowledge acquisition need each other.
> Knowledge acquisition produces the wrong stuff.
> Machine learning destroys the right stuff.
> Fuzzy, Fuzzy [illustration of a peach], Crunch, Crunch [illustration of cereal].
> KA needs more crunching, ML needs some more understanding.

From the above tropes, we can determine that knowledge is not only tangible but *stuff*. *Stuff* is defined "as raw material, the material or substance out of which anything is or can be made." This definition would predict that things can be made out of knowledge. (We will return to this constructive aspect of knowledge in step four below.) Also from the above slogans, we learn that there can be different kinds of knowledge (i.e., *right, wrong, fuzzy, and crunchy*). The

knowledge from knowledge acquisition is fuzzy, softer (KA is described as generating a lot of fuzziness on p. 8) and the knowledge from ML is harder.

To describe the link between tools/techniques and knowledge, the following trope was a helpful summary:

> . . . development of tools that were specialized for the acquisition, representation, and interpretation of knowledge for one problem-solving method. Experience has showed that such hard-coded tools are brittle and always require tweaking when used for a new application."(vol. 2, no. 2, p. 8)

From this point, we can generalize further about the properties of knowledge and tool/techniques in knowledge acquisition:

- Knowledge is substance and tools can be used on it.
- Knowledge has varying degrees of hardness or softness. Knowledge which is in numerical form is harder. Knowledge which is not amenable to numerical representation, such as understanding, is softer and fuzzier.
- Tools share the properties of the knowledge they work on and the results from the tools' use are prone to the disadvantages of the substances that the tools were used on. Thus, *hard-coded tools are brittle* and soft *KA generates fuzziness.*

In sum then, tools, techniques, and knowledge share the property of hardness and softness. We can generalize about hardness and softness as follows. First, hardness seems to refer to the more tangible or operational, whereas soft refers to more intangible, like understanding, or conceptual. Additionally, hardness refers to how computationally tractable something is (i.e., numbers are hard and computers are number crunchers). Second, the hardness/softness is relative. For instance, the physical computer equipment is called hardware; the programming used by the equipment is called software. However, of the two activities involving programming, Machine Learning (ML) is described as harder than Knowledge Acquisition (KA) which is referred to as being fuzzier or softer than ML. Third, the hardness or softness of a tool or technique determines what it can be used on. Thus, soft KA techniques tend to be used on qualitative knowledge and harder number crunching tools or algorithms work on quantitative information. This reasoning differs from common-sense Western thinking where we match tools to materials according to function (e.g., a hammer to drive in a nail, a needle to sew cloth . . .) so some tools are hard, like a needle, and used on soft materials,like cloth, and vice versa. Fourth, tools have weaknesses that follow from their hardness or softness. For example, hard-coded tools are described as brittle, not easily applied to other applications. KA (which is soft) lacks and needs more (hard) number crunching capability, ML (which is harder) needs some more understanding (softness or fuzziness).

In step four, we search for tropes which would support the prediction (from the definition of *stuff*) that the knowledge is . . . "material out of which anything . . . can be made." We look for verbs and nouns relating to *making* or *building* and find numerous ones. We group them by subject (underlined) as follows:

<u>blocks, frameworks and structures</u>
structures
frameworks
building blocks for problem solving structures (vol. 2. no. 2, p. 8)
levels and higher order descriptions mentioned

<u>levels or layers</u>
KADS four-layer model
building meta-level knowledge acquisition tools (vol. 3. no. 1. p. 5)
high-level representations (vol. 3. no. 1. p. 5)
many systems are attempting to reuse knowledge at different levels to gain leverage during knowledge acquisition (vol. 3, no. 1, p. 6)

models

acquiring and modeling appropriate problem-solving knowledge.

construction of knowledge relates to models

different kinds of models (e.g. data-base vs. k-base)

models can have layers

Clancey claimed that every knowledge-based system contained a blackboard architecture model. . . . (vol. 3, no. 1, p. 5)

Katharina Morik (vol. 2. no. 2, p. 8) stated that the construction (or modeling) view sees knowledge acquisition as a constructive process during which the knowledge is created and represented in the knowledge base. Katharina stated the opinion that only the construction view is a viable one and that machine learning techniques actively create the model of the knowledge.

Brian Woodward (vol. 2, no. 2, p. 10) stated that knowledge only exists in combination with humans, and that the knowledge engineer's understanding is needed to build a model.

systems

(vol. 2, no. 2, p. 10) Hans Akkermans sees KE as just another branch of engineering and he draws the interesting analogy between knowledge and energy: you cannot grasp either of them, but you can use them to engineer systems.

From the above quotes, we can verify that knowledge is used in building (e.g., *knowledge is created, KA is a constructive process*, and so on). To determine what is being built, we need to know how to order the *blocks, levels, models,* and *systems*; that is, determine what is being used to build what, and learn what the final construction is. The following quote is useful in this ordering:

> "an expert must understand the task at hand, a knowledge engineer must learn about the task and understand its notation in a conceptual model. A software engineer must understand the model and be able to transform it into a system" (vol 2. no. 2., p. 9)

Thus, the system is the final construction and it contains models, according to the above quote and that of Clancey's. Models, in turn, are described as being used, like tools, for problem solving.

4.1.2. Metaphoric claim that knowledge is a human construction, an artifact

From the above semantic analysis of metaphor, we arrive at the metaphoric claim that *knowledge is a human construction.* In the KA Newsletters, there was a noticeable emphasis on the use of tools and the building of problem-solving structures, frameworks, models, or systems. This emphasis leads us to believe that the metaphoric claim—*knowledge is a human construction*—could be restated to *knowledge is a human tool-using and -building activity.* That is, knowledge acquisition is seen as a tool/technique borrowing, tool/technique creating activity that leads to the construction of something (a system) which will be a greater problem-solving tool. We could, in turn, simplify either of the above claims to *knowledge is an artifact,* given the common Western definition of artifact as something made by human work.

4.1.3. Comparison of metaphoric analyses of knowledge acquisition and anthropology

At the most general level, the knowledge acquisition text included mainly mechanical metaphor while that of the anthropological text (according to Salmond), natural metaphor. Let us consider some of the properties of the mechanical versus the natural metaphor. Some properties of *mechanical* are that it is human made for some purpose, it is tool and/or labor intensive, it can break or need adjustment, and it is a recent development in the human time span. On the other hand, *nature* is viewed as of natural or divine creation, unalterable, and old in the geological time span (which is much older than the human time span).

We would like to speculate on the origin of the human-made and natural type of metaphor in knowledge acquisition and anthropology. We believe that the use of these two different metaphors relate to the history of these fields. For instance, the field of anthropology started before the use of computers. Additionally, anthropology involved studies of societies which were less technologically oriented, that were closer to nature, than western society. In contrast, knowledge

acquisition developed as a branch of computer science, artificial intelligence, after computers were being used to solve complex problems. Furthermore, the purpose of knowledge acquisition is to provide knowledge bases which will be used in expert or knowledge-based computer systems. Thus, knowledge acquisition comes from a machine-oriented field and its oriented toward machines in its purpose.

Thus, the origins of these two fields may help in explaining the differences in their metaphors. These differences are summarized as follows. In knowledge acquisition, the focus is a tool-intensive activity that results in a human-made construction. In anthropology, the emphasis is on understanding, either from the larger view of the structure and functioning of the society (often in comparison to other societies) or from the close-range focus on the individuals and the meaning of their lives as seen through their eyes. Thus, in knowledge acquisition the descriptive verbs are often *building* or *constructing*, whereas in anthropology, they are *seeing* or *revealing* the natural, pre-existing landscape. Little mention is made of tools in anthropology, perhaps, because fieldwork methods are not tool intensive. Indeed, the keeping of observations and thoughts in field journals was and still is the main instrument of anthropology. Furthermore, fieldwork is described as "data gathering, as though the important features of another society will be lying about on the ground for our collection (Salmond, 1982, p. 76) Thus, it is not surprising that so little reference to tool metaphors is made in anthropological text. The occasional references to *building* in anthropology are reserved for the building of theoretical systems[5], sometimes described as edifices or monuments on top of the landscape (thus, providing a new or better vantage point [Salmond, 1982, p. 78]. In contrast, in knowledge acquisition, theory is a component of the tools/techniques and used in creating the physical system. The system, in turn, does not provide the vantage point for the viewing of something else, as it does in anthropology. This distinction can be seen in the use of *levels, layers, and height* in the two fields. In the KA newsletter, the levels or layers are those of those of models of problem solving (e.g., *models have layers, the relation of the knowledge level with the computational level* [vol. 2, no. 2, p. 8]) and refer to points of interest in the construction; in the anthropological examples they refer to vantage points for viewing the natural landscape.

4.2. SAMPLE ANALYSIS AND CLASSIFICATION OF THE ETHNOMETHODOLOGICAL APPROACH

4.2.1 Application of semantic analysis of metaphor to ethnomethodology text

We chose to analyze the ethnomethodology approach because it has recently been applied to the areas of software engineering and knowledge acquisition. Ethnomethodology, a sociologically based-approach, aims to describe what people actual do, often in the participants' own words (Kwasnik, 1990, p. 33). McIlvenny and Raudaskoski (1990, p. 32) describe it as the "study of . . . the body of common-sense knowledge and range of procedures and considerations by means of which the ordinary members of society make sense of, find their way about in, and act on the circumstances in which they find themselves. (Heritage, 1984, p.4)." Ethnomethodology usually involves highly detailed naturalistic studies of interaction and conversation, such as are done in Conversation Analysis.

Two current texts were used for the analysis: the *Proceedings from the AAAI Workshop on Complex Systems, Ethnomethodology and Interaction Analysis* (Gilbert, 1990) and *Computers and Conversation* (Luff, Gilbert & Frohlich, 1990). Instead of tracing through our semantic analysis of ethnomethodological metaphor, we will summarize our findings and focus on comparing the ethnomethodological approach to knowledge acquisition and on classifying the former.

---

[5] More frequently, theoretical activity in anthropology is described in terms of the land, such as when particular theoretical approaches *bear fruit.* (Salmond, 1982, p. 77)

4.2.2. Metaphoric claim that knowledge is rooted in social activities
The metaphoric claim that knowledge is rooted in social activities came from the following quote:

> Knowledge is an emergent property of social agreement. . . in the sense that knowledge is rooted in social action, in ways of life and in patterns of activity. . . . seeing that knowledge is rooted in social action as opposed to knowledge being a negotiated, emergent product of social action will allow us to better understand the role of agreement in the materials we examine. (Button, Sharrock & Anderson, 1990, p.11)

Button seems to be making a distinction that knowledge is not created just by people agreeing to regard something as true but by the underlying means by which people reach agreement (i.e., patterns of social activity). Social activities in ethnomethodology take place within a context and the principles by which behaviors are organized vary according to context. Kwasnik (1990, p. 124) defines context as:

> that part of a stream of experience which envelopes a person and which we call that person's situation. It includes the person's self, the things in the environment, and the factors which provide the background against which that person creates meaning for him or herself.

To paraphrase then, knowledge is rooted in social activities and social activities take place within and are organized by a context. Hence, *knowledge is rooted in context.*

4.2.3. Comparison of metaphoric analyses of ethnomethodology and knowledge acquisition
To review, we have proposed that ethnomethodology's metaphoric claim simplifies to *knowledge is rooted in context*, whereas that of knowledge acquisition's leads to *knowledge is an artifact.* As in the comparison of anthropology and knowledge acquisition, ethnomethodology has naturally based metaphors (e.g., *rooted, embedded, stream of experience, activities*), and knowledge acquisition, mechanical ones. We would speculate that ethnomethodology has natural types of metaphors because it comes from a similar tradition to anthropology, although more recent. Conversation Analysis, for example, came from the sociology of the 1960's. It is in keeping with their respective backgrounds that ethnomethodology portrays knowledge as being *embedded in the patterns of social activities* while knowledge acquisition describes it as being *a tool for constructing greater tools.*

When construction terms are used in ethnomethodology, they have a different sense than when they are used in knowledge acquisition. The former's construction terms are, for example, *backdrop* and *scaffolding of facts to apply concepts* (Button et al, 1990, p. 12). These terms are defined in the dictionary as being temporary supports and of occurring in the theater. In contrast to the temporary quality of the construction terms, the terms of *context* and *situation* have a very solid, grounded sense because of their association with the verbs, *rooted* and *embedded.*

Perhaps one of the most noticeable differences between metaphor in ethnomethodology and knowledge acquisition lies in their treatment of the persons studied. In the knowledge acquisition text, we found very little mention of experts in comparison to the knowledge engineer. A much more common term which encompassed the expert, as well as information from text, was *knowledge source* (vol. 3. no. 1. p. 19). If we might speculate on the use of *knowledge source,* we would propose that it emphasizes the importance of the existence of a source over the attributes of the source commonly considered important in Western culture (e.g., human-nonhuman and animate-inanimate).

In the ethnomethodology texts, a number of articles described those being studied as *actors* or *agents.* We would propose that these terms emphasize action or behavior rather than thought or the individual. Indeed, Sacks, one of the key developers of Conversation Analysis, admonishes practitioners not to worry about what or how people are thinking but to "Look to see how it is that persons go about producing what they do produce" (Wooffitt, 1990, p. 13). In other words, in ethnomethodology, persons are seen in terms of their behaviors as these arise from situations, whereas in knowledge acquisition they are considered in terms of their contribution to the knowledge-building enterprise.

Ethnomethodology also differs from knowledge acquisition in its terms for computers when computers are among those studied. Ethnomethodology has lent itself to the study of human-machine interactions because of its focus on behaviors, on interaction, and on naturalistic study. In ethnomethodological texts where computers are studied, we noticed a blurring of the common Western distinction between person and machine. For instance, in the AAAI Workshop (1990, p. 5), ethnomethodology and Conversation Analysis are described as being applied to "the dialogue and interaction between 'intelligent agents', both complex systems and human beings." Many of the other terms used for those studied, such as *collaborating agents*, *partners in conversation*, and *entities*, could refer to either humans or machines. In addition, some texts attribute to computers capabilities and traits that have been associated with humans in Western science (e.g. the capability to participate in discourse or to do participant observation and the trait of being socially disadvantaged). For instance, Shaw and Gaine's article (1990, p. 58) focuses "on discourse between collaborating agents [human and computer] with differing specialties in a domain. . . . the computer system should act as a 'participant observer', with all that this entails in both involvement in the discourse process and analysis of the values introduced through observation." In addition, the title of Durham's paper, *Why Computers Are Socially Disadvantaged* (Luff et al, 1990, p. 45) attributes a trait to machines (in interaction with people) that sociologists have traditionally assigned to certain people.

We have focused on ethnomethodology and knowledge acquisition's terms for those studied because how people are characterized in research can have a profound effect on the description of behavior. Secord (1990) analyzed the ways in which social scientists have characterized human participants in experiments and other types of research. As he traced the shift from characterizing participants as *subjects* to characterizing them as *persons*, he showed how this has had a profound effect on the description of behavior, the choice of research methodology, and the nature of theory employed. For example, language associated with *subjects* was very much that of behaviorism with the scientist as the one who provides the description of what is going on (usually in terms of changes in environmental inputs [stimuli] or changes in internal [organismic] states). Language associated with *persons* is more concerned with participants who can initiate actions, exhibit purposeful behavior and provide a description of what is going on. As such, participant behavior cannot simply be read off by a naive observer and experimenters use their knowledge of the world to describe human action.

In ethnomethodology, researchers are likely to use their own knowledge to describe other persons' actions because, as experiencers of the same situation, they would have the same insight. We would argue that because the individual's knowledge and behavior stems from the situation, the individual could be considered an *artifact* of the situation. Indeed, Gaines (1991) may be describing this possibility in his statement: "The ontogenetic bias of both behaviorism and cognitive science is being challenged by the ethnomethodological perspectives in which the very notion of an 'individual' is an experimental artifact. . ."

4.2.4. Classification of ethnomethodological approach as structural or task-based

Using the classification scheme proposed in section 3.2, we would propose that ethnomethodology is a task-based approach. The ethnomethodological approach focuses on behaviors as these occur in context. Thus, ethnomethodology has the focus on observables of a task-based approach. We would further note that many of the verb metaphors used in CA use the visual sense (e.g., the Sack's "*Look* to *see* how it is that persons go about producing what they do produce" quote earlier mentioned) as would be expected with a focus on observables.

We would propose that ethnomethodology is a task-based approach for another reason—its aim to to describe the underlying organization, rules or patterns by which behavior are produced. This aim links the ethnomethodology approach to Salmond's structural approach, the precursor to the task-based classification.

Using the task-based classification of the ethnomethodological approach, we can then speculate as to some of the weaknesses it might have, especially as applied to the field of knowledge acquisition. As a field that focuses on observables, ethnomethodology is unlikely to be able to contribute theories that would produce models with explanatory or causal capabilities (Harré, 1986; Paton et al, 1991b; Rothbart, 1991). This is because there are few scientific theories which are used to describe entities in the real world that are dependent solely on observables.

5. Discussion of the Uses of Metaphoric Analyses

We propose that metaphoric analyses can show us how how our understanding and practice of knowledge acquisition is limited or constrained. For instance, the view of knowledge as a construction/tool/artifact could warn us that we are taking a construction, or engineering view, to the detriment of an more investigative one. Emphasis on knowledge as a construction can have two meanings in knowledge acquisition: (1) that knowledge bases are constructions, and (2) that concepts are constructed. The products of the construction process can be seen or viewed. However, as Snell remarked (section 2.1), there is more to knowledge than recognition and know-how. Understanding is also required. The understanding of a domain comes from an investigative rather than engineering process. In this respect it is, or should be, the scientific activity from which the engineering develops.

In addition, semantic analyses of metaphor can also show us our cultural constraints in understanding knowledge. The sciences have developed through discourse and the discourse of science comes from a Western tradition. Thus, when anthropologists or knowledge engineers are working with experts or users from different cultures they will be ethnocentric in their communication, even when they are trying to *merge epistemic horizons* with the expert. Salmond (1982, p. 86) offers little hope of cultural relativism for she notes that individuals from different scholastic/cultural traditions will not understand knowledge in the same way. However, she does propose that semantic analyses may, at least, allow individuals of different cultures a means of understanding one another.

In light of Salmond's proposal, we would urge that metaphoric analyses be applied to knowledge engineer's interviews of the experts or to recordings of inter-expert meetings. Paton has used metaphoric analyses, for instance, to characterize aspects of the domain by studying the language an expert used (Paton et al, 1991b). In more detail, such metaphoric analyses involve the knowledge engineer's attempts to learn the expert's view of the domain and its boundaries by eliciting the history of the domain, its major metaphors, and theories. The information received in each interviewing session guides and refines the questioning in the next. From, the interviewing sessions, the knowledge engineer develops models on paper (which include such aspects as the vocabulary of discourse, types of metaphors, the relationships between them, and relations between the domains) to characterize the domain (Paton et al, 1990a, p. 14).

Metaphoric analyses could also be employed to discover shifts in conceptualization of important concepts in the field. For example, numerous authors in the knowledge acquisition text (such as, Breuker and Morik, Newsletter, vol. 2. no. 2, p. 8-10) commented on the change from the view of knowledge being transferred or extracted from the experts to one where it was constructed or modeled. The metaphors regarding this view of knowledge acquisition could be analyzed in previous and current texts to learn when the shift occurred, what other tropes changed, and what else was occurring in the field that might have contributed to this change.

Semantic analysis could also serve as one means for determining whether a combination of knowledge acquisition techniques were based on similar or conflicting views of knowledge. Shaw and Woodward (1989) have identified a potential danger in knowledge engineer's tendency to pick methods from other fields to meet an application's needs—the result being a patchwork of methods with possibly conflicting assumptions. While, they sought to determine the method's underlying assumptions about the nature of knowledge by examining the knowledge representation (e.g., they

cite the example of entity-attribute grids which assume that distinctions are primitives underlying knowledge [1989, p. 29-2]), we would propose that analyses of metaphor might also serve. For example, our analyses of metaphor in ethnomethodology and knowledge acquisition (section 4.2.3) showed that they have fundamentally different views of the nature of knowledge. In ethnomethodology, knowledge is viewed as arising from social situations, whereas in knowledge acquisition, from tool construction.

6. Summary

Our aim in this paper was to introduce the technique of semantic analysis of metaphor for use in examining knowledge acquisition text. Analysis of metaphor allows us to go beyond our tacit understanding of the field to gain new insights into how we, as practitioners, view knowledge and the role of the knowledge engineer and knowledge source. Also in this paper, we proposed that different approaches to knowledge acquisition could be classified according to their use of metaphor as task, concept, or domain based. We illustrated with the classification of the ethnomethodological approach how this classification could help its user anticipate the applications and problems of a particular approach.

We mentioned the four following uses of the proposed method of metaphoric analysis and classification:
to learn how our view of knowledge acquisition is constrained, culturally, or otherwise by our use of metaphor;
to discover shifts in the conceptualization of key concepts in the field;
to characterize the experts' understanding of the domain; and
to anticipate whether different approaches are based on similar or conflicting views of knowledge.

We are indebted to Trevor Bench-Capon, Hyacinth Nwana and Michael Shave, co-workers of Ray on the MEKAS project. Thanks also go to Patricia McKay for her insightful comments on this paper and to Rom Harré for his advice regarding aspects of section 2. In addition, we would like to thank those who reviewed this paper for KAW'91 for their valuable comments.

References

Arbib, M. A. (1972). *The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory*, New York: John Wiley.

Arbib, M. A. (1989). *The Metaphorical Brain 2: Neural Networks and Beyond*, New York: John Wiley.

Aronson, J. L. (1984). *A Realist Philosophy of Science*, London: Macmillan.

Aronson, J. L. (1991). Verisimilitude and type hierarchies. *Philosophical Topics*. In press.

Boose, J. & Gaines, B., Eds. (1990). *Knowledge Acquisition for Knowledge-Based Systems Newsletter*, 3, 1, February.

Boose, J. & Gaines, B., Eds. (1990). *Knowledge Acquisition for Knowledge-Based Systems Newsletter*, 2, 2, May.

Boyd, R. (1979). Metaphor and theory change: What is 'metaphor' a metaphor for? In Ortony, A., Ed., *Metaphor and Thought*, pp. 356-408. Cambridge: Cambridge University Press.

Button, G., Sharrock, W. & Anderson, B. (1990). Talking through the soft-ware design, *Proceedings from the AAAI Workshop on Complex Systems, Ethnomethodology and Interaction Analysis*, Boston, Massachusetts, July, pp. 9-20.

Clancey, W. J. (1989). The knowledge level re-interpreted: modelling how systems interact. *Machine Learning*, 4, 285-91.

Forsythe, D. E. & Buchanan, B. G. (1989). Knowledge acquisition for expert systems: Some pitfalls and suggestions. *IEEE Transactions on Systems, Man, and Cybernetics*, 19, 435-42.

Gaines, B. R. (1991). Between neuron, culture and logic: explicating the cognitive nexus. Abstract of paper to be presented at the International Conference on Cognitive Science: Tools for the Development of Organisations, Montreal, Canada, April 29-May 2.

Gaines, B. R. (1989). Social and cognitive processes in knowledge acquisition. *Knowledge Acquisition*, **1**, 39-58.

Genter, D. & Grudin, G. (1985). The evolution of mental metaphors in psychology: a ninety year perspective. *American Psychologist*, **40**, 181-92.

Gilbert, N., Ed. (1990). *Proceedings from the AAAI Workshop on Complex Systems, Ethnomethodology and Interaction Analysis*, Boston, Massachusetts, July.

Harré, R. (1970). *The Principles of Scientific Thinking*, London: Macmillan.

Harré, R. (1986). *Varieties of Realism*, Oxford: Blackwell.

Harré, R. (1990). Exploring the Human Umwelt. In Bhaskar, R., Ed., *Harré and His Critics*, pp. **. Oxford: Blackwell.

Heritage. J. (1984). *Garfinkel and Ethnomethodology*, Cambridge: Polity Press.

Hunter, D. E. & Whitten, P., Eds. (1976). *Encyclopedia of Anthropology*, New York, New York: Harper & Row.

Johnson, M. J., Ed. (1981). *Philosophical Perspectives on Metaphor*, Minneapolis: University of Minnesota Press.

Koch, S. & Deetz, S. (1981). Metaphor analysis of social reality in organisations. *Journal of Applied Communications Research*, **9**, 1-15.

Kwasnik, B. H. (1990). An analysis by means of naturalistic approaches of two complex behaviors, *Proceedings from the AAAI Workshop on Complex Systems, Ethnomethodology and Interaction Analysis*, Boston, Massachusetts, July, pp. 123-35.

Lakoff, G. & Johnson, M. (1980). *Metaphors We Live By*, Chicago: Chicago University Press.

Lewis, C. S. (1939). Bluspels and Flalansferes. In Lewis, C. S., Ed., *Rehabilitations and other Essays*, pp. 133-58. Oxford: Oxford University Press.

Luff, P., Gilbert, N. & Frohlich, D., Eds. (1990). *Computers and Conversation*, Computers and People Series, London, England: Academic Press.

McIlvenny, P. B. & Raudaskoski, P. (1990). The practicing machine or the machine in human practice? *Proceedings from the AAAI Workshop on Complex Systems, Ethnomethodology and Interaction Analysis*, Boston, Massachusetts, July, pp. 31-40.

Meyer, M., Picard, R. & Ross, J. (1990). The ethnological method of participant observation as a technique for manual knowledge acquisition. *Proceedings of the 5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November, pp. 21-1 to 21-15.

Morgan, G. (1980). Paradigms, metaphors and puzzle solving in organisation theory. *Administrative Science Quarterly*, **25**, 605-22.

Newell, A. (1982). The Knowledge Level. *Artificial Intelligence*, **18**, 87-127.

Ortony, A. (1975). Why metaphors are necessary and not just nice. *Educational Theory*, **25**, 45-53.

Paton, R. C. (1988). An investigation into the use and meaning of 'system' in biological education, Unpublished Ph.D. Thesis, The University of Leeds.

Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1990a). Foundations of a structured approach to domain characterization. MEKAS report No. 13, November 24, Department of Computer Science, University of Liverpool, Liverpool L69 3BX, U. K. .

Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1990b). Towards a classification of approaches to knowledge acquisition: A proposal. MEKAS report No. 15, December 8, Department of Computer Science, University of Liverpool, Liverpool L69 3BX, U.K.

Paton, R.C. (1991). Towards a metaphorical biology, Working paper CS/1991/21, Department of Computer Science, University of Liverpool, Liverpool, L69 3BX, U.K.

Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1991a). Transfer of natural metaphors to parallel problem solving applications. In Schwefel, H-P. & Maenner, R., Eds., *Parallel Problem Solving from Nature*, Lecture Notes in Computer Science, Berlin: Springer. In press.

Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1991b). From real world problems to domain characterisations. *Proceedings of the 5th European Knowledge Acquisition Workshop*, Crieff, Scotland, May 20-24. In press.

Paton, R. C., Nwana, H. S., Shave, M. J. R. & Bench-Capon, T. J. M. (1991c). Systemic metaphor analysis for knowledge based system development. To appear in Jackson, M.C., et al, Eds, *Systems Thinking in Europe*, London: Plenum.

Robinson, H. (1990). Towards a sociology of human-computer interaction: A software engineer's perspective. In Luff, Gilbert, and Frohlich, Eds., *Computers in Conversation*, pp. 39-50. London: Academic Press.

Ricoeur, P. (1971). The model of the text: Meaningful action considered as text. *Social Research*, **38**, 529-62.

Ricoeur, P. (1973). Creativity in Language. *Philosophy Today*, **17**, 97-111.

Ricoeur, P. (1981). *Hermeneutics and the Human Sciences*, Cambridge: Cambridge University Press.

Rothbart, D. (1984). The semantics of metaphor and the structure of science. *Philosophy of Science*, **51**, 595-615.

Rothbart, D. (1991). Metaphors, models and meanings in science, paper to be presented at a seminar on metaphor. Director Professor Rom Harré, Oxford University, June 6th.

Soskice, J. M. (1985). *Metaphor and Religious Language*, Oxford: Clarendon Press.

Salmond, A. (1982). Theoretical landscapes: On cross-cultural conceptions of knowledge. In D. Parkin, Ed., *Semantic Anthropology*, ASA Monograph 22, pp. 66-87. London: Academic Press.

Secord, P.F. (1990). 'Subject' versus 'persons' in Social Psychological Research", in Bhaskar, R., Ed., *Harré and His Critics*, pp. 165-88. Oxford: Blackwell.

Shaw, M. L. G & Gaines, B. R. (1990). Computer support of a conversation-theoretic methodology. *Proceedings from the AAAI Workshop on Complex Systems, Ethnomethodology and Interaction Analysis*, Boston, Massachusetts, July, pp. 57-68.

Shaw, M. L. G. & Woodward, J. B. (1989). Mental models in the knowledge acquisition process. *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October, pp. 29-1 to 29-24.

Snell, B. (1953). *The Discovery of the Mind: The Greek Origins of European Thought*, Oxford: Basil Blackwell.

Way, E. (1991). *Metaphor and Knowledge Representation*, Dordrecht, Holland: Kluwer.

Wooffitt, R. (1990). On the analysis of interaction. In Luff, Gilbert & Frohlich, Eds., *Computers in Conversation*, pp. 7-38. London: Academic Press.

Psychotherapeutic Techniques for Identifying Experts' and System Users' Sensorial Experience of Thinking: Significance to Knowledge Acquisition

MARY A. MEYER* AND JEFFREY M. ROSS+
MS F600, Statistics Group, Los Alamos National Laboratory
Los Alamos, New Mexico 87545, USA
*email: myym@lanl.gov

+*Los Alamos Medical Center, Los Alamos, New Mexico 87544, USA*

Abstract

According to some psychotherapists, humans experience their thinking as a mental making of pictures, hearing of internal dialogue, and/or feeling of corporal states. We propose that knowledge engineers could better establish rapport and acquire knowledge from the experts if they identified the sensory forms that the experts used. Additionally, knowledge engineers could anticipate some of the difficulties that users encounter in using a knowledge-based system, if they identified the pattern of sensorial representations that the system required. This paper describes how to identify individuals' sensorial representations and apply these psychotherapeutic techniques to situations arising in knowledge acquisition.

1. Introduction

People internally represent and evaluate their thinking processes by making mental pictures (visual), hearing dialogues (audial), and/or feeling of corporal position, movement, or tension (kinesthetic). That is, when people contemplate some future possibility or try to recall some information, they will use audial, visual, or kinesthetic representations. For instance, when people are trying to remember the spelling of a word, they may say it to themselves, picture it in writing, and/or evaluate their proposed spelling by internally checking their bodies' response to see if the spelling *feels right*. We will term peoples' tendency to represent information in visual, auditory, or kinesthetic forms, as their sensorial experience of thinking, or SET. When an individual uses multiple sensorial forms in combination, such as in the above spelling example (e.g., a sequence of audial, visual, and kinesthetic), we will refer to this as a SET sequence. We will term the SET that the individual naturally uses to begin thinking or communicating as his or her preferred SET. For instance, in the spelling example, the individual uses the audial SET first.

Some psychotherapists are able to infer the clients' SETs from aspects of their nonverbal (direction of gaze, facial expression, and gesture) and verbal communication (use of metaphors and choice of verbs). After identifying the SET the client prefers, the psychotherapist can emulate the client's SET as a means of promoting understanding and rapport with the client. In addition, the psychotherapist can use the clients' SETs to help them effect changes in their behavior. For instance, when individuals are thinking about performing some task that they have had difficulty doing, such as figuring their taxes, their SETs provide them with negative visual, audial, and

kinesthetic feedback which discourages them from beginning the task (Meyer & Ross, 1991). The psychotherapist may assist the client in creating a different SET, a different representational process, one which encourages the client to perform the activity.

In this paper, we make explicit the means by which the psychotherapist discerns the client's SET. We offer this information to knowledge engineers for their work with source experts and system users because we believe that there is an analogy between therapists, knowledge engineers, and computer systems and between clients, source experts, and users of computer systems. Just as the therapist utilizes the clients' SETs to communicate and effect change, we suggest that the knowledge engineers may acquire knowledge more quickly, directly, and effectively by attending to the variations in the experts' SETs. In addition, we propose that knowledge engineers will find the source experts' and system users' SETs useful in developing models of their thinking. For instance, a panel of psychiatrists and psychologists, who reviewed the usefulness of this psychotherapeutic technique for the U.S. Army, mentioned its potential in modeling expert performance, such as in marksmanship (Swets & Bjork, 1990, p. 90). We also propose that the knowledge engineers could develop models of the user's SETs as these are structured by the user's interaction with the knowledge-based system. In this way, the knowledge engineer would anticipate where the users will encounter SET-related difficulties in the use of the system and how to resolve them. Similarly, we believe that intelligent computer systems, in the future, could better engage their users' attention and participation if they were more responsive to the users' SET preferences and sequences.

This paper is organized as follows. In section 2, we present background information on the internal sensorial process and the theory behind it. In section 3, two manual techniques are given for identifying experts' SETs: stimulated response and guided questioning. In addition, we suggest options to allow knowledge engineers to match their preferred SETs to those of the experts or users. We also outline a means for investigating the SET functioning that a knowledge-based system may implicitly require of its users. For each of these techniques, we describe the knowledge acquisition situation in which they would be used and how they would be administered. In section 4, we discuss the implications of peoples' sensorial thinking to knowledge acquisition. Section 5 is a summary.

2. Theoretical Background

2.1. HISTORY

2.1.1. Bandler and Grinder's Neuro-Linguistic Programming

The concepts presented in this paper have their origins in the seventies when (1975, 1976 & 1979), Bandler, a mathematician, and Grinder, a linguist, introduced the neuro-linguistic programming model. While much of their model is borrowed from linguistics and cognitive psychology, they introduced the very useful concept of people thinking in terms of picture making, dialogue, or kinesthetics.

Einspruch and Forman (1985), reviewers of the literature on Neuro-Linguistic Programming, or NLP, describe it in the following way. NLP is concerned with the way in which people process information. According to the NLP model, people process information in three modes: visual, auditory, and kinesthetic. The sensory modes that individuals use and the sequence in which they use them are believed to be critical to their performance of particular tasks. That is, persons who

are expert at particular tasks have discernibly different processing sequences than those who perform poorly on the same task. The modes by which the expert processes information can be determined, codified, and taught to others. How individuals process information is determined by observing their eye gazing and linguistic patterns. Bandler and Grinder demonstrated the NLP model in the therapeutic arena. They learned the sensory patterns used by noted psychotherapists, such as Milton Erickson, Virginia Satir, and Fritz Perls, in effecting change in their clients and codified them in a way that allowed these psychotherapists' skills to be transferred to others interested in learning them.

While there has been some questioning, such as by Sharpely (1987)[1], of Bandler and Grinder's claims of the therapeutic efficacy of their NLP model, we believe that it is a useful tool for sensorially mapping individuals' internal cognitive processes. Perhaps, the origins of NLP, linguistics and cognitive psychology, suit it to identifying peoples representational processes from their communications.

2.2. THE BASIS OF NEURO-LINGUISTIC PROGRAMMING IN COGNITIVE THERAPY

The NLP approach, in general, and our application of it to knowledge acquisition can be viewed as belonging to the field of cognitive psychology. Their relationship to cognitive psychology is significant because a form of it, cognitive therapy, is one of the few psychological therapies that is supported by the data and considered to be a trainable technique.

While the idea of peoples' thoughts influencing their emotions is over 2500 years-old, Aaron Beck is generally credited with originating cognitive therapy in the 1950's (Burns, 1980, p. 10). Beck developed cognitive therapy within the context of treating clinical depression.

There are three principles of cognitive therapy: The first principle is that all of a person's emotions are created by her or his cognitions. The relationship of NLP to cognitive therapy becomes clear when a *cognition* is defined:

> A cognition refers to the way that you *look* at things—your perceptions, mental attitudes, and beliefs. It includes the way that you interpret things—what you *say* about something or someone to yourself.[2] (Burns, 1980, p.11)

Thus, according to the first principle, people feel the way that they do because of their thoughts, in particular, the way that they *look* at things and what they *say* to themselves. In NLP-based therapies, this principle is used to help clients effect change in their lives, such as to overcome fears and phobias. Because the clients' sensorial way of thinking has lead to their demoralization and inability to change, the therapist changes the clients' sensorial encoding (e.g., replacing the unpleasant feedback from the SET or changing the SET types).

The second principle is that when people are feeling depressed, their thoughts are strongly negative, and the third principle is that these negative thoughts almost always contain gross distortions. The third principle has important therapeutic applications. In therapy, the client's cognitive distortions are identified and shown to be twisted thinking. For instance, in Burn's *Feeling Good: The New Mood Therapy*, one of the 10 cognitive distortions is called *all-or-nothing thinking*. This cognitive distortion is defined as occurring when individuals see everything in

---

[1] Sharpley notes (1987, p. 106), that NLP is neither magical nor new, rather it is a compendium of procedures beneficial to counseling.

[2] Italics are ours.

black and white; thus, if they perform less than perfectly, they label themselves as failures. In NLP, the individuals' cognitive distortions are identified using a linguistic analysis of the surface structure of their communication (For further details on this complex linguistic approach, see Grinder and Bandler [1976]).

2.3. HOW PSYCHOTHERAPEUTIC TECHNIQUES CAN BENEFIT KNOWLEDGE ENGINEERING

First, psychotherapy can provide interactionally-based techniques, such as NLP, for identifying patterns in peoples' behavior and communication that relate to their thought processes. For instance, Bandler and Grinder have proposed: that when peoples' gaze goes up as they are thinking, they are visually processing; when their gaze moves horizontally back and forth, they are aurally processing; and when they gaze down, they are kinesthetically processing. Similarly, people will use verbs and metaphors in describing their thoughts that indicate the sensorial process they used. For instance, the verb and metaphor usage in the following statement, in combination with a downward gaze, would indicate that the person was using a kinesthetic processing: *I feel that the deadline will be too tight.* These techniques will be discussed further in sections 3.1 and 3.3.

Second, the psychotherapeutic technique of matching (also called mirroring) the individual's verbal and nonverbal behavior is generally accepted as an effective means of establishing rapport in counseling (Sharpley, 1987, p. 104). In addition, we have previously argued (see Meyer, Picard & Ross, 1990, p. 21-4) that the practice of matching the individual's expression, breathing, and posture is a means of gaining insight into the client's feelings. The technique of matching is described further in section 3.2.

Third, psychotherapeutic techniques exist for assisting people change their SETs when these have cause them to become stuck and demoralized. While, it may not be appropriate for knowledge engineers to change experts' or users' SETs, we would propose that they could use the psychotherapeutic techniques specifically to learn where users are likely to become *stuck* in using the knowledge-based system. Suggestions on how to identify the usage of SETs that the system implicitly requires is given in section 3.4.

3. The Psychotherapeutic Techniques For Identifying SETS

3.1. IDENTIFICATION OF THE SET TYPE THE INDIVIDUAL IS USING

Identifying which of the SET types the individual is referencing is necessary for performing the other techniques, such as determining the individual's sequencing of SETs, matching the expert's preferred SET in eliciting and representing knowledge, and learning what SET demands the knowledge-based system may place on the user. Thus, the knowledge engineer would seek to identify the individual's SET, if s/he intended to use any of the other techniques mentioned in this paper. In addition, we would recommend that the knowledge engineer identify expert's or user's SETs, if s/he was interested in developing models of their thought processes.

3.1.1. Stimulated response technique

The stimulated response technique is so named because the therapist begins with questions that are likely to cause the client to think in particular SET so that the client's variations in sensorial representation can be noted. For example:

What's the color of your front door? stimulates the client to use a visual process,
What are the lyrics or melody to a particular song? stimulates an audial processing, and
How did you feel when you had the flu? stimulates a kinesthetic process.

Note these questions can be adjusted to the situation, such as the individual's area of expertise or interest.

After asking questions which favor the use of a particular SET type, the therapist observes the individual's direction of gaze and gestures. While common indicators of the uses of the different SETs are given in table 1 below, individuals may have their own idiosyncratic indicators. For instance, the knowledge engineer may observe that the expert widens her eyes and tips her head to the right when internally dialoguing or otherwise trying to aurally remember information. In the future, then, when the expert exhibits these same nonverbal behaviors, the knowledge engineer can infer that she is using an audial SET.

Therapists may not need to ask questions like those above to decode their client's verbal and nonverbal communications but may simply note, over time, how the individual uniquely signals, verbally and nonverbally, which SET he/she is using. However, we list the stimulated response technique because knowledge engineers are unlikely to have the luxury of regularly meeting with the experts or users through time as therapists commonly do with their clients.

TABLE 1
Common indicators of the three types of SETs

SET type	Direction of eye gaze	Gesture	Verbal communication
Visual	up or unfocused and straight forward.	turning head to look away from other person. drawing pictures with hands.	Visually-related verbs and metaphors, like *see, show, picture, clear,* or *perspective.*
Audial	side to side.	cocking head, as if listening.	Aurally-related verbs and metaphors, like *hear, sound,* or *say.*
Kinesthetic	down and/or to right and down.	whole body slant, hand to heart or gut with facial expression, and/or doing something tactile, like rubbing one's forehead or pencil.	Verbs and metaphors relating to feeling, touch, movement, or smell.

3.1.2. Guided questioning technique

With guided questioning the therapist simply asks the question, *How do you know what you know?* (after the individual has responded to a previous question) and supplies the following options: *Are you making a picture? Are you telling yourself in words? Do you have a feeling?* The therapist then watches the client's eye movements and body language to assist the client and, perhaps, supplement the client's answer. For instance, if the client was not sure how she was

sensorially representing, the therapist might ask whether or not she was using the SET that her eye motions had indicated. If, on the other hand, the client responded that she was talking to herself, the therapist could add his observations, such as, *I thought you were also making pictures.* The client might agree and explain that she saw the words and then read them to herself.

The guided questioning technique could be used, if the therapist were unsure of how the client were sensorially processing in a particular instance and wanted to verify his/her reading of the client's verbal and nonverbal indicators. In general, the therapist would only use this technique if s/he had already introduced client to idea of sensorial experiencing of thought.

Preliminary to guided questioning. One way of introducing individuals to the concept that they process information through different sources is to take them through the experience. We provide the following example.

Therapist: I will ask you a question when I signal with my hand (lowers his hand as if to signal the start of a race) and you will try to answer it. The question will be about something you have not been thinking of to help us learn how you come to know. (The therapist signals with his hand.) What is your name?
Client: Jane.
Therapist: How did you know? (The therapist watches for indicators given in table 1 above, as the client tries to reconstruct how she knew her name and remember which SET she used.)
Client: I saw it like it was written on a blackboard, or
I heard my name called, or
I felt it in my bones.

The question may have to be asked several times, and the questioner should explain to individual that *knowing how one knows* often take some time. If the individual is unable to say how s/he knew but has exhibited clear indicators of one of the types, the therapist could ask if it was a particular type. For instance, if the individual appeared as if s/he were listening to a Sony Walkman and looked side to side, the therapist could ask *were you hearing your name*? Those individuals who favor visual processing, such as graphic artists, are usually quickest to know how they know; those favoring audial processing are next, and those favoring kinesthetic processing, are the last.[3] Often the individual is surprised that s/he is receiving the information from these different sources. However, after most persons have been acquainted with the idea, they can answer the therapist or knowledge engineer's guided questioning on how they know what they know.

3.2. THE MATCHING OF THE INDIVIDUAL'S SET

3.2.1. Why the knowledge engineer would wish to match the expert's or user's SETs

As mentioned earlier, the two reasons for matching an individual's SET were (1) to enhance rapport, and (2) to enhance understanding of his/her (e.g., the client's, expert's, or user's) thinking. Grinder and Bandler (1986, p. 14) state that "the most important result of comprehending your client's representational system is trust." They argue that matching the client's SETs with verbs and metaphor of the same type allows the client to understand the therapist's communication with greater ease and creates the beginnings of trust. The client believes that the therapist understands him and that, since the therapist is capable of understanding him, he is also capable of helping him. Because of the client's trust, then, the client is willing to follow the

[3] Literature on teaching and negotiating has described the percentages of the general population who prefer to receive information visually as 45%, audially as 45%, and kinesthetically as 10%.

therapist's lead. (The client's willingness to follow the therapist's lead is the clinical definition for rapport.)

We add that the exercise of matching the individual's SET will increase the knowledge engineer's understanding of the individual as only empathy can. In a previous paper on the anthropological field method of participant observation (Meyer, Picard & Ross, 1990), we described the technique of trying to approximate others' experiences as a means of understanding others' views in their own terms.

3.2.2. Determination of the knowledge engineer's preferred SET

A helpful preliminary to matching is for the therapist or knowledge engineer to become aware of the SET that s/he prefers. The preferred SET is typically the one that a person starts a thinking process or a SET sequence with.

Exercise for learning preferred SET. The exercise which allows the reader to identify his or her preferred SET is as follows.

Imagine yourself taking a vacation at the beach. . . . What is the first thing that you thought of? Were you picturing, hearing, or feeling this aspect of the beach?

This simple exercise is effective in determining an individual's preferred SET but a similar one could work equally well. If the reader wishes to formulate another exercise, s/he should take care to make the description sense neutral. In other words, the exercise should *not* focus on a situation that invokes one sense over another, such as eating which would particularly invoke the kinesthetic. Similarly, the instructions in the exercise should *not* direct the listener's choice of SET. For instance, the instructions, *Picture yourself at the beach*, would direct the listener to use the visual SET.

At the very least, the exercise given above or a similar one will the make knowledge engineer aware of how s/he will naturally tend to ask the expert or user to think. That is, in communicating with the expert or user, the knowledge engineer is likely to select topics, questions, and methods and to describe things in terms of his/her preferred SET. It is also helpful to know the knowledge engineer's preferred SET for matching, as described in section 3.2.3.

3.2.3. Matching options

One option for matching the knowledge engineer's and the expert's/users' SET preferences is to select the knowledge engineers or the experts/users accordingly. After the expert's/user's and knowledge engineer's sensory preferences have been determined, the knowledge engineer(s) can be paired with experts/users of the same sensory preferences. The knowledge engineer will have determined the expert's/user's preferred SET by identifying his/her SET sequences in section 3.3. The knowledge engineer will have learned which SET s/he prefers by performing the exercise in section 3.2.2. Often, this option of matching knowledge engineers to experts or users will not be possible because there will not be enough knowledge engineers, experts, or users.

In most cases, then, the knowledge engineer will need to do as the therapist does, identify the expert's or user's SET preference, and match it. In Grinder and Bandler (1986, p. 15), there is a table of how a therapist would communicate the same message using different verbs to match a client whose preferred sensory representation was visual, auditory, or kinesthetic. For instance,

for the therapist to communicate that he did not understand the client, the therapist might say the following:

I don't see what you're saying, to a client with a visual preference.
I don't think I'm hearing you clearly, to a client with an audial preference.
or *I don't feel I'm getting what you're saying*, to a client with a kinesthetic preference.

3.3. IDENTIFICATION OF THE INDIVIDUAL'S SEQUENCING OF SETS

3.3.1. Description of sequencing of SETs

People may use several SETs: their preferred or lead SET to start a thinking process, some others, and one which finishes the sequence. For instance, in solving a simple summing problem, the individual may first visualize the numbers, say the solution, and then check its correctness by picturing the numbers equaling the solution. This SET sequence would be visual, audial, and visual, and the preferred SET would be visual.

3.3.2. Reasons for identifying the individual's sequencing of SETs

One reason that knowledge engineers or therapists may wish to identify SET sequences is that they give the history of the individual's thinking which has resulted in the individual's verbal response. That is, they provide the context of the individual's final thoughts and/or response on the matter.

Another reason for focusing on the sequence is that each SET type gives different qualities, nuances, to the thought (Meyer & Ross, 1991). For instance, when a person uses the visual SET, there is often the nuance of the *larger picture*. That is, when people are considering doing some task, their visualization may be like a wide angle view which shows them doing the task or depicts the complete, finished product of their efforts.

In contrast, the audial SET is frequently narrower with a play-by-play, sportscaster-like emphasis of the individual's activities. People may hear themselves thinking about the next step they need to do or whether they have remembered something correctly. In this way, the audial SET serves as a place marker for where the individual is in a mental process, such as for a librarian doing card cataloguing. Then too, the audial SET frequently has a critical or prescriptive tone. For instance, people often hear internal voices commenting negatively on their ability to do some task or their progress so far (e.g., *you really should do this task*, or *you will never finish on time*). In this case, the audial SET can have the tone of an domineering person or an adult telling a child what to do.

On the other hand, the kinesthetic SET has the flavor of evaluating or checking something. For example, Bandler and Grinder propose that good spellers often do a kinesthetic check for whether a pictured spelling *feels* right. Then, too, people may do a kinesthetic check of how they feel about something they've represented with the other types of SET (e.g., *would I feel good about that?* or *does that feel right*?).

The therapist or knowledge engineer is able tease out each SET type's contributions to the other's meaning. In sum, noticing the sequencing allows the knowledge engineer/therapist to bring out the subtleties of the thought or the layers of meaning that would otherwise be missed.

3.3.3. Techniques for identifying the individual's sequencing

To identify sequencing, the therapist or knowledge engineer asks a question and then notes the indicators of the SETs as they punctuate the speaker's phrases. If the therapist or knowledge engineer has a question as to which SET type the individual is referencing, s/he can use the guided questioning technique.

The following conversation illustrates how noting the sequencing of SETs can add dimensions of meaning to a response as simple as remembering what one had for dinner.

Therapist: What did you eat last night?
Client: Ice cream (kinesthetic gaze) but I can't remember what else.
Client: I wasn't *feeling* good (kinesthetic gaze) but I told myself I had to eat something (audial gaze) or I would get sick.
Oh yeah, I actually fixed a dinner (visual gaze).

The above sequence was kinesthetic, kinesthetic, audial, and visual. On the basis of the sequence, the therapist was able to hypothesize about about the client's experience of thinking. As a check, the therapist brought the sequence to the client's attention, and the client was able to explain her experience of thinking.

Client: I could remember the ice cream because I felt *stuffed* , so it was a kinesthetic SET. Then, I was able to remember that I had earlier not *felt* well. I remember *telling* myself that I'd better eat something. And then, I remembered the first part of the dinner because I *saw* the meal in the pan on the stove.

The client's explanation verified the therapist's understanding of client's thinking. In particular, note that client's audial SET had a prescriptive nuance and that the visual SET depicted the finished product, the prepared meal.

Another option would be for the knowledge-based system to match the users' preferred SET. Currently, most computer systems seem to use visual representations. Thus, we would expect the best matches of SETs to be among experts/users with visual preferences. For instance, if experts were going to be taught to use the prototype knowledge-based system as a means of entering and checking the expert knowledge, it might be prudent to select experts who preferred visual SETs.

In the future, it may be possible to build intelligent computer systems that have the capabilities to identify and match their users' SET preferences and sequences, just as therapists identify those of their clients. This capability would allow improve the users' perception of the rapport in the interaction and belief that system could help them. We would expect that such user interfaces would better engage their users' attention and participation than those without this capability.

3.4. IDENTIFICATION OF THE SET USAGE THAT THE SYSTEM REQUIRES OF ITS USERS

3.4.1. Reason for identifying the SET usage required by the knowledge-based system

One reason for attempting to identify the usage of SETs required by the knowledge-based system is that it may require the users to do awkward sensorial processing. We have observed, for instance, that internally processing in one SET type makes it difficult to externally receive information simultaneously in this same type. For example, when people take notes as they listen, they often talk to themselves, especially if they are not recording verbatim but are interpreting the information. If someone else demands that they listen, they may have difficulty because they are

already using the audial SET type. In this situation, people may try to drown out the distracting incoming audial by making their internal dialogue louder (i.e., by talking to themselves aloud).

We have observed another example where the individual's SET sequencing can be disrupted. When watercolorists are engrossed in developing the design (values, negative and positive space, and shapes) of a painting, they will talk to themselves, often audibly. We would propose that their visual channel for thinking is totally engaged, and, thus, to help themselves in problem solving, they resort to the audial. Thus, we would expect that their work would be disrupted if someone talked to them. Evidence that this is the case came from a watercolor instructor's advice to her class. She cautioned the class against listening to the classical music radio channel while they were painting because *the announcer could break in at a critical point and cause a disaster.*

It is interesting that a few watercolorists, by my informal polling,[4] could listen to cappella music without disrupting this process. The exception was in the case of vocalists singing in languages that were not understood by the watercolorist. For instance, one person could listen to Gregorian chants while painting and said that it helped her *block out other distractions.*

In the use of some computer programs, we have encountered similar problems. For example in using the Table capability in Microsoft Word, we try to visualize what our table will look like. While we are picturing our table, we have to do something else visual, remember which pull-down menu the table format command that we wish to use is in. Because the table format commands are located in three different pull-down menus, *Edit, Format,* and *Document,* we generally must pull down each and look for the command. We are unable to picture the menu containing the command while still retaining in our mind's eye how we wish the table to look.

3.4.2. Proposed techniques for identifying SET usage caused by interaction with the knowledge-based system

We suggest that knowledge engineers, newly sensitized to SET sequences in humans, use the systems that they are creating and note their own SET responses as a means of checking for the problems noted above. That is, the knowledge engineers could try to learn whether, as users, they must simultaneously employ the same sensorial type for thinking as for receiving information. Knowledge engineers might also find it of interest to examine the patterns of SETs evoked by programs that users consider easy to use versus those that they consider difficult.

In addition, we recommend that the knowledge engineers take someone who is unfamiliar with the system through the *how do you know what you know* exercise (Section 3.1.2.) and have him/her check for difficult SET processes in using the system. Later, a few users could be asked to follow this procedure to identify remaining problems with the system's SET usage or sequencing.

4. Discussion

We propose that attention to the individual's SETs is important for the following reasons. First, we think that it would be helpful to the knowledge engineer's modeling if, at least, they were aware that the expert/user had this sensorial process going on. To date, knowledge-based and expert systems may include computational models of the expert's problem-solving behavior but they do not, to our knowledge, reference the sensorial representations by which the expert solved

[4] Personal communications with Mary Stovall's watercoloring class of 1989.

the problem. A natural place for the inclusion of the expert's SETs might lie in intelligent tutoring systems. The literature reviewing NLP (Einspruch & Forman, 1985; Swets & Bjork, 1990) suggests that experts have a different sequence of SETs than nonexperts and that expertise can be modeled with reference to these SETs. We wonder if tutoring systems could include descriptions of the expert's SET sequence, if only to motivate the students. That is, the expert's sequence of SETs are likely to contain positive messages for doing the particular activity and the student is more likely to desire to do the activity, if s/he has been lead, in an exercise, to adopt the expert's SET as his or her own (Meyer & Ross, 1991). In addition, we would propose that an intelligent tutoring system could use SETs, in much the way that therapists do, to change the student's way of perceiving something, particularly if they are having problems with the lesson.

Second, we believe that it is important that the nontechnical issues of software engineering be included with technical. Robinson (1990), a software engineer, has argued that the human-computer interface is being marginalized within engineering and being placed outside of any context. The context, in this case, would be the interaction between entities where some natural identification and matching of SETs occurs. Currently, human users "attempt to collaborate on a successful dialogue and adapt the style of their queries to the kind of responses produced by the other participants", whether these participants are other humans or computers (Brennan, 1991, p. 59-60). We are proposing here that the analogy of psychotherapist and client be extended to the computer system and user, that systems be developed with the capabilities to identify and match the user's SETs (perhaps through monitoring individual's viewing patterns[5]). We expect that the computer systems could better engage their users' attention and participation if they had these capabilities.

5. Summary

Our aim in this paper was to provide knowledge engineers with guidance on how to identify and use the SET preferences and sequences of source experts and system users. To this end, we have tried to make explicit the means by which psychotherapists discern clients' internal sensorial processing. Techniques for identifying knowledge engineer's, expert's, and user's SET preferences were given. We also described options for how the knowledge engineer could match the expert or user's SETs. In addition, we outlined a plan for how knowledge engineers can learn of the system's influence on the user's SET patterns.

We believe that knowledge engineers would better engage the attention and cooperation of experts and users by attending to variations in their sensorial processing. In addition, we propose that knowledge-based systems in the future have the capability of identifying users' preferred SETs and matching the users' preferences in the ways that they present instructions and information.

We are indepted to the anonymous reviewers of this paper for their valuable comments.

[5] "Crosby and Stelovsky demonstrated the utility of unobtrusively monitoring and analyzing duration of user's eye fixation on different areas of the program". . . .They "conjectured that although individual viewing patterns (eye fixation) do not appear to be related to comprehension, they may be indicators of user preferences about the system's response strategy" (Brennan, 1991, p. 58).

References

Bandler, R. & Grinder, J. (1975). *The Structure of Magic: A Book About Language and Therapy*, Palo Alto, California: Science and Behavior Books.

Bandler, R. & Grinder, J. (1975). *Frogs into Princes: Neuro Linguistic Programming*, Moab, Utah: Real People Press.

Burns, D. D. (1980). *Feeling Good: The New Mood Therapy*, New York, New York: William Morrow and Co.

Carberry, S. (1991). Second international workshop on user modeling. *AI Magazine*, special issue, **11**, 57-60.

Einspruch, E. & Forman, B. (1985). Observations concerning research literature on neuro-linguistic programming. *Journal of Counseling Psychology*, **32**, 589-96.

Grinder, J. & Bandler, R. (1976). *The Structure of Magic II: A Book About Communication and Change*, Palo Alto, California: Science and Behavior Books.

Keeney, B. & Ross, J. (1985). *Mind in Therapy: Constructing Systemic Family Therapies*, New York, New York: Basic Books.

Meyer, M. A. , Picard, R. R. & Ross, J. M. (1990). The ethnological method of participant observation as a technique for manual knowledge acquisition, (LAUR-90-1654), *Proceedings from the 5th Knowledge Acquisition for Knowledge-Based Systems Workshop*, November, Banff, Canada, pp. 21-1-21-15.

Meyer, M. A. & Ross, J. M. (1991). *How To Enjoy What You Don't But Wish You Did: Dealing With Everyday Procrastination*, unpublished manuscript.

Musen, M. (1989). Conceptual models of interactive knowledge acquisition tools. *Knowledge Acquisition*, **1**, 73-88.

Sharpley, C. (1987). Research findings on neurolinguistic programming: Nonsupportive data or untestable theory? *Journal of Counseling Psychology*, **34**, 103-7.

Swets, J. A. & Bjork, R. A. (1990). Enhancing human performance: An evaluation of 'New Age' techniques considered by the U.S. Army. *Psychological Science*, **1**, 85-96.

Robinson, H. (1990). Towards a sociology of human-computer interaction. In Luff, Gilbert & Frolich, Eds., *Computers and Conversation*, pp. 39-49. London, United Kingdom: Academic Press.

Extracting logical knowledge from prescriptive texts in order to build deontic knowledge bases

Bernard Moulin and Daniel Rousseau
Département d'informatique, Université Laval,
Ste-Foy, Québec G1K 7P4, Canada
tel: (418) 656-5580 ou 656-7979,
Email : MOULIN@LAVALVM1.BITNET

Abstract

In project ACAT (Acquisition des Connaissances et Analyse de Textes) we aim at elaborating knowledge bases after analysing the information contained in prescriptive texts (regulation texts for instance). We have developed a knowledge acquisition system, ACAT, which identifies the logical structure of prescriptive texts from which it generates a set of rules, called "deontic rules".
In this paper we study the problem of the logical interpretation of cross-references that are heavily used in regulation texts. We present a model which enables our system to detect these references in regulation texts. We describe the characteristics of a system which creates a knowledge base, whose "deontic rules" are equivalent to the logical content of the original text. We show how knowledge structures derived from cross-references are used for generating deontic rules.

1 INTRODUCTION

Texts contain large amounts of knowledge from which people extract relevant information in order to carry out their activities. For any literate person, it seems that acquiring knowledge from texts is a straightforward process : you read the text, identify whatever information is relevant for your purposes, and, you memorize it! That is easy and natural for human beings! However, it is a very difficult task when we think to implement it in a software : its complexity is equivalent to that of building natural language undertanding systems.
In order to build practical systems, several researchers have chosen a strategy which avoids the burden of understanding completely texts, by extracting only specific knowledge structures which are relevant for their applications.
Some projects aim at extracting lexical knowledge from texts (such as dictionaries) and at generating lexicon entries which could be used by natural language parsers (Zernick 1989, Jacobs 1988, Velardi et al. 1989).
Other systems like RESEARCHER (Lebowitz 1988), analyse narrative texts and generalize the recognized knowledge structures (physical object descriptions, events) in generalized

schemata. The WIT system (Reimer 1990) aims at extracting from technical texts terminological knowledge (properties of concepts and of relations between concepts): it acquires new concept representations, using inductive generalization.
Another promising area corresponds to the extraction of knowledge in order to index texts. G. Boy (1990) presents an approach to acquire and refine text indices according to the context : his system can help people to access efficiently relevant information within large bodies of documentation.

In project ACAT (Acquisition de Connaissances et Analyse de Textes), we have worked on the problem of extracting logical knowledge from prescriptive texts (law, regulation normative texts). We have developed a knowledge acquisition system, SACD (Moulin and Rousseau 1989, 1990-1 to 3), which identifies the logical structure of prescriptive texts, from which it generates a knowledge base composed of a set of so called "deontic rules".
A prescriptive text is composed of three types of propositions (Gardies 1980):

- sentences which introduce definitions of terms which are relevant for the application domain;
- "normative propositions" which describe prescriptions and contain modalities (terms indicating permission, obligation, interdiction);
- "metatextual statements" which are not related to the domain itself, but give various information about the text : they correspond to cross-references from one part of the text to another one.

Definitions can provide interpretation problems but have no influence on the text logical content which is expressed by normative propositions and metatextual statements.
In order to study the logical content of normative propositions, we can use modal logic, and, more specifically, deontic logic[1] when it comes to regulation texts (Kalinowski 1972). According to Kalinowski's suggestions, we work with texts composed of "normative propositions" whose syntactic structure characterizes norms : they are built with verbs or expressions indicating a modality such as an obligation ("devoir"; *must or ought to*) a permission ("pouvoir"; *to be allowed or may*) or an interdiction ("il est interdit de"; *it is forbidden to*).

Metatextual statements can greatly influence the logical content of prescriptive texts. The use of cross-references makes difficult the interpretation of legal texts (Tremblay 1987) (Lagacé 1987) (Bench-Capon et al. 1987), especially when there are chains of references introduced by expressions such as "in spite of article x", "subject to provisions of article y" etc. Cross-references are used by legal writers to ease their task of formulating concise sentences, but a proliferation of references puts a heavy burden on the reader when s/he tries to interpret a prescriptive text : the reader must undertand and make explicit the chains of references in order to infer what are the precise conditions which influence the application of text prescriptions.

Our system SACD analyses prescriptive texts and create knowledge structures on the basis of the logical information which is extracted from normative propositions and metatextual statements. When generating its deontic knowledge base, SACD makes explicit the application conditions of the text prescriptions : we obtain "deontic rules" which may be used to validate the coherence and complexity of prescriptive texts, and can become the core of a knowledge base for a consultation system.

In section 2 we present briefly the model that has been developed for the analysis of prescriptive texts. We present in more details the structures which model cross-references. In section 3 we describe the approach that is used to generate the rule base which reflects the text logical content. In section 4 we give examples of deontic rules generation.
The examples which are presented in this paper have been taken from a regulation text on taxi drivers (Gouvernement du Québec 1987). In order to simplify our presentation, we have changed the numbers of articles from the original text.

2. ANALYSIS OF PRESCRIPTIVE TEXTS

2.1 macrostructure and microstructure

We distinguish in a regulatory text three types of components (Moulin 1988): the text macrostructure which corresponds to the information dealing with text presentation (headings, titles, organization of the text in chapters, sections, articles etc.); the text microstructure which organizes the logical content of the text (keywords or expressions introducing conditions, exceptions, modalities, references etc.); the domanial component of the text corresponding to the information which is specific of the application domain and belongs neither to the macrostructure nor to the microstructure.
Analysing the text microstructure enables us to identify the text component parts which characterize the text logical content : relevant domain objects and their related prescriptions, définitions, normative propositions and their application contexts, references. Here, we present briefly the main elements of a text microstructure (for more details see Moulin and Rousseau 1990-1 and -2).

We call a **modal operator** any expression which points out the modality in the sentence. The modal operator characterizes normative propositions (Kalinowski 1972). the three main modalities are obligation, interdiction and permission. **An operator scope** is the part of the sentence on which the modal operator is applied. A scope that precedes a modal operator is called front-scope, whereas, the back-scope corresponds to a scope which follows the operator.
In figure 1 we present the parse proposed by SACD after analysing the microstructure elements of article 1 :

A [taxi driver] ..... *(front scope of the modal operator "cannot")*
< cannot> ..... *modal operator*

decline to run an errand whose origin is situated
within the township or region for which
the taxi license has been delivered, ... *(back scope)*
unless that ..... *(exception connector)*
the destination of the errand goes beyond the outer limits
of this township or region by more than 50 kilometers. ..... *(scope of the connector)*

The modal operator plays an important role within a normative sentence : it helps identifying the object (usually situated in the front scope and emphasized with square brackets by SACD) on which holds the prescription (usually situated in the back scope): that's the case for most regulation texts from the Gouvernment of Quebec (Tremblay et al. 1984). In our example, the prescription is attached to the object "taxi driver".
Classical **logical operators** are also identified in the text : "et" *("and")*, "ou" *("or")*, "ne ... pas" *("not")*, "ne ... plus" *("no ... more")*, etc.

A connector is a conjunction which introduces a **condition** ("lorsque", "si" *("when", "if")*) or an **exception** ("sauf si", "à moins que" *("except", "unless")*). A connector may be situated either in a front scope or in a back scope of a modal operator. A connector is followed by a scope which is different from the modal operator scopes.
References in prescriptive texts correspond to other microstructure elements which are detected by SACD. They are presented in detail in section 2.2.

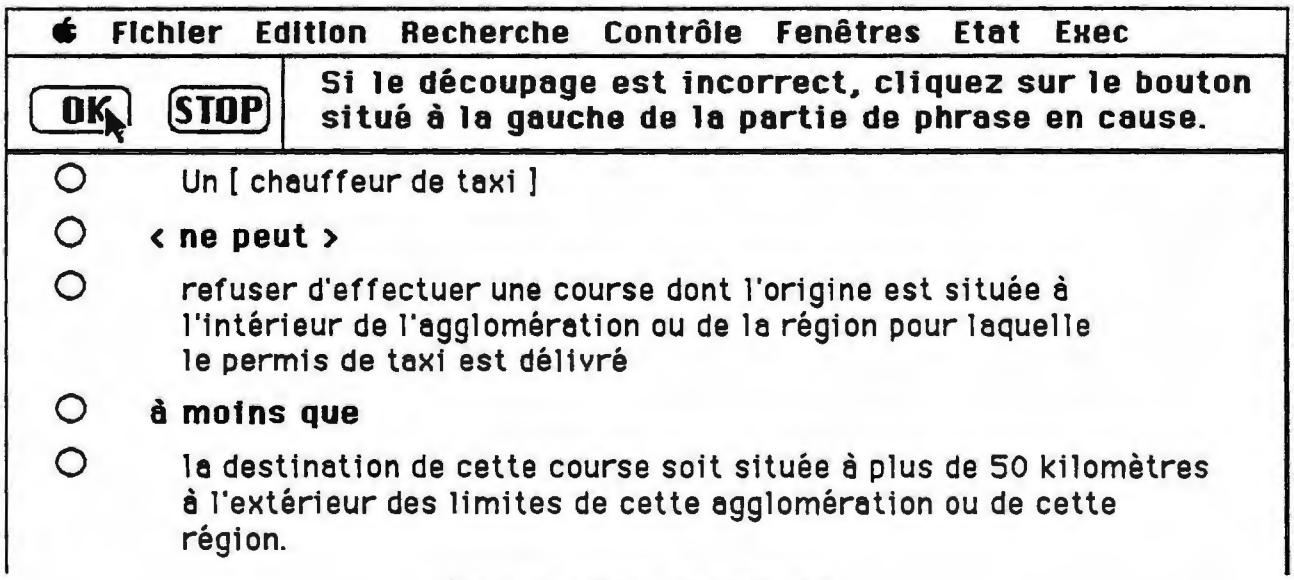


Figure 1 : Parse for article 1

As our objective is to identify the components that support the logical content of a prescriptive

text, it is not necessary to analyse the text at a semantic level. For efficiency sake, our approach consists in analysing syntactically prescriptive texts, using two text grammars in order to analyse successively the text macrostructure and microstructure (Moulin and Rousseau 1990-1 to -3). SACD presents the results of the parse to the domain specialist for validation. If the parse is correct, the specialist clicks on the OK button. If not, s/he clicks on the button facing the line where there is an error (see figure 1). SACD presents a menu which contains the various types of errors that can be detected according to the category of the component that has been indicated by the specialist. S/he chooses the error type and SACD presents a specific interface for correcting this error in the parse (for more details see Moulin and Rousseau 1990-1).
Errors in the parse may be the result of wrong formulations in the original text; and the text correction fixes the error. Errors may be caused also because SACD text grammars are not rich enough to cope with some new microstructure element in this particular text. In this case, the knowledge engineer analyses the problem and modifies SACD text grammars accordingly. Hence, the grammars are improved with each new text analysed by SACD. Currently the grammars are fairly stable and require few modifications.

After having completed the microstructure analysis, SACD helps the specialist to organize in a type hierarchy, the objects that have been detected in the text. For instance "taxi driver" is categorized as a sub-type of "physical person", "taxi" as a sub-type of "vehicule" etc. The type hierarchy is the back-bone of the deontic knowledge base to be constructed by SACD.
In the last phase of the text analysis, SACD identifies anaphoric references such as personal pronouns, possessive or demonstrative adjectives and some adverbs which are used in the text to refer to an object within the same sentence or in another one. For example in the sentence "when a taxi driver receives the payment for an errand, he must give back the exact change to the passenger", "he" is an anaphora which refers to "taxi driver". Anaphoras are the most common type of references encountered in texts. Human are good at solving anaphoric references (finding the object which is referred by the anaphora) and they do it naturally. But anaphora resolution is done on the basis of a semantic and often pragmatic analysis of the text. SACD detects the various types of anaphoras at a syntactic level. Then it prompts them to the specialist and asks her/him to indicate what is the object referred by the anaphora. This is a practical and efficient way of solving anaphoras, even if the process can be a little tedious for the specialist. But afterwards, the text is completely analysed and all relevant domain objects have been detected. (Moulin et Rousseau 1990-1). The next phase consists in generating the deontic knowledge base (DKB) which will contain a set of rules equivalent to the text logical content. This phase is described in detail in section 3.

2.2 Internal references

Several microstructure constructs are used to indicate references from one portion of the text to another. We call them **internal references** to distinguish them from "external references". The latter refer to portions of texts which are different from the text in which they appear. We

present here the main categories of internal references we have identified in our text corpus. This categorization is important because the logical processing of the DKB rules is done with respect to the internal reference category (see sections 3 and 4).

References introduced by a "referential connector"

Those references are introduced by connectors like "malgré" ("in spite of"), "nonobstant" ("notwithstanding"), "sous réserve de" ("subject to"), etc. and correspond to expression types like the following :

- **{Malgré, Nonobstant} [les dispositions de] {l'article n, l'alinéa n}**
  X op.mod P1 **[si** contexte1]
- ***{In spite of, notwithstanding} [provisions of] {article n, sentence n}***
  ***X mod.op** P1 [if context1]*
- **Sous réserve des dispositions des articles n1 à n2,**
  X op.mod P1 **[si** contexte1]
- ***Subject to provisions of articles n1 to n2,***
  ***X op.mod** P1 [if context1]*
- **{Conformément à, outre} l'article n,** X **op.mod** P1 **[si** contexte1].
- **{In conformity with, with respect to} article n,** X **mod.op** P1 **[if** context1].

We are using the following conventions :
"mod.op" indicates the sentence modal operator, "if context" corresponds to the conditions for applying the prescription; X represents the object on which bears the prescription; when one element is optionnal, it is presented between square parenthesis; when several elements may be used, they are presented between braces.

Referential connectors introduce an explicit reference to one or several articles (or sections or sentences). Figure 2 shows the analysis of article 2. SACD identifies the referential connector "malgré" ("in spite of") and the associated internal reference "article 1" :

In spite of ..... *referential connector*
<<article 1>>, ... *internal reference*
a [taxi driver] who does not hold the first place in a waiting station,
<must>
refuse to take the customer who wants to embark, and tell him to ask the first taxi in the row.

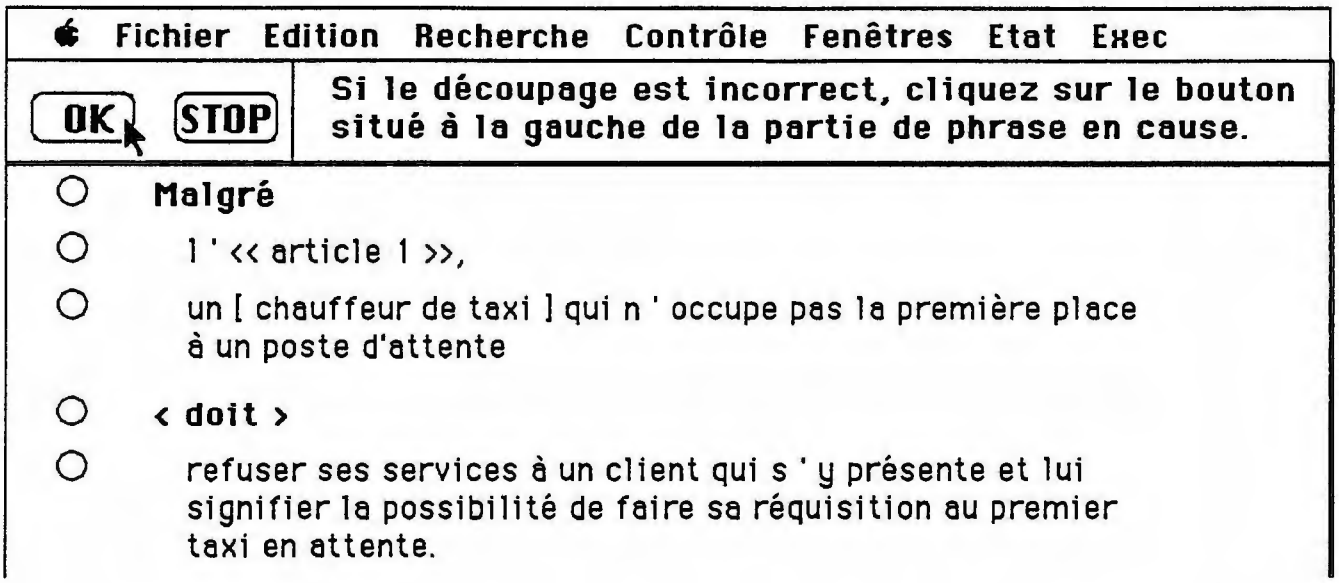


Figure 2 : Parse of the referential connector"malgré" (article 2)

Interstatement connectors

Interstatement connectors introduce an internal reference referring to the previous sentence. Let's consider the following cases :

- positive interstatement connectors such as "de plus" ("in addition"), "en outre" ("besides") etc.

X **op.mod1** P1 **si** contexte1. **De plus,** X **op.mod2** P2 **si** contexte2

*X **mod.op1** P1 **if** context1. **In addition,** X **mod.op2** P2 **if** context2*

- negative interstatement connectors such as "toutefois" ("nevertheless"), "cependant" ("however") etc.

X **op.mod1** P1 **si** contexte1. **Néanmoins,** X **op.mod2** P2 **si** contexte2.

*X **mod.op1** P1 **if** context1. **However,** X **mod.op2** P2 **if** context2*

- the connector "dans ce cas" ("in this case")

X **op.mod1** P1 **si** contexte1 et **si** contexte2. **Dans ce cas,** X **op.mod2** P2.

X **mod.op1** P1 **if** context1 and **if** context2. **In this case,** X **mod.op2** P2.

The connector "in this case" has a special behaviour : it applies on the last context ennunciated in the preceding sentence and its logical processing differs according to the fact that the referred context is positive or negative. Figures 3 and 4 present a negative case.

Article 11 sentence 1:

When

a taxi driver receives the payment for an errand,

he

<must>
give back the exact change to the passenger
unless ..... *exception connector*
the payment for the errand is done with a bill ..... *introducing a negative context*
surpassing the errand fare by more than 20 $

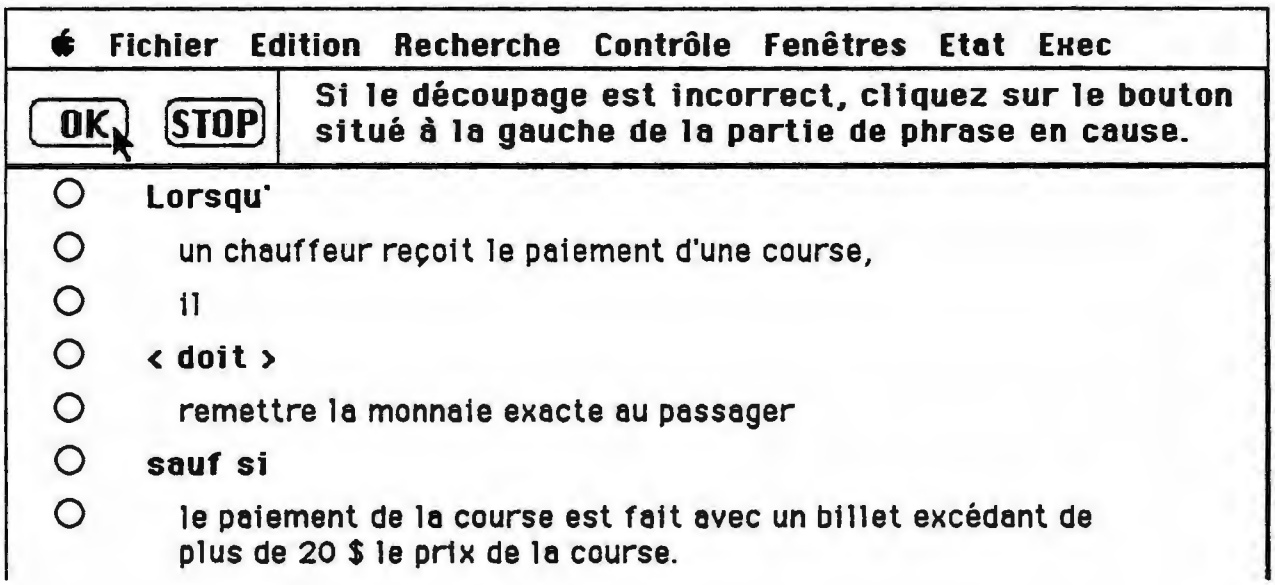


Figure 3 : Article 11 sentence 1

Article 11 sentence 2:
In this case,
the [taxi driver]
<is not obliged to>
accept the payment for the errand.

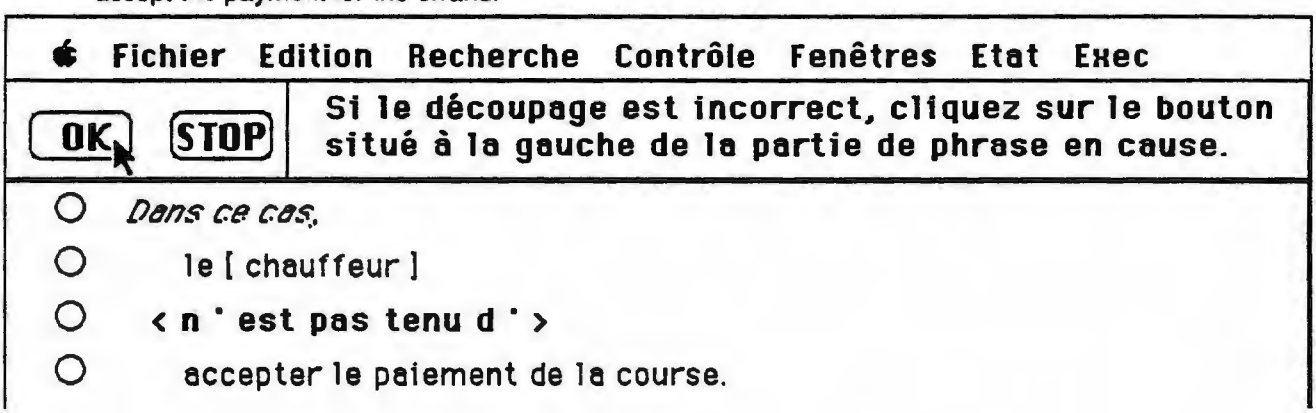


Figure 4 : article 11 sentence 2

Referential expressions

They indicate a reference between an object and one or several articles in which this object has been defined. We distinguish three categories.

- referential expressions of conformity such as:
  - ... un X **{prescrit par, requis par, conforme à} l'article n**
  - ... *an X {prescribed by, required by, conforming to} article n*
- referential expressions of exclusion such as:
  - ... un X **exclu par l'article n**
  - ... *an X excluded by article n*
- referential expressions of quotation such as:
  - ... an X **{visé à, décrit à} l'article n**
  - ... *an X described in article n*

Metatextual statements of applicability

These internal references indicate explicitly that some text portions apply or do not apply on other objects than those that are mentionned in the referred text portions.

- Positive metatextual statements of applicability
  - . **{la présente section, le présent chapitre, le présent article} s'applique [mutatis mutandis] aux** {X,[Y],[Z]}
  - . *{the current section, the current chapter, the current article} applies on {X,[Y],[Z]}*
  - . Les {X,[Y],[Z]} **sont soumis aux exigences énoncées à {l'article n, la section s}**
  - . *{X,[Y],[Z]} are subject to the provisions of {article n, section s}*
- Negative metatextual statements of applicability
  - . **{la présente section, le présent chapitre, le présent article} ne s'applique pas aux** {X,[Y],[Z]}
  - . *{the current section, the current chapter, the current article} does not apply on {X,[Y],[Z]}*
  - . Les {X,[Y],[Z]} **ne sont pas soumis aux exigences énoncées à {l'article n, la section s}**
  - . *{X,[Y],[Z]} are not subject to the provisions of {article n, section s}*
- Restrictive metatextual statements of applicability
  - . **{la présente section, le présent chapitre, le présent article} s'applique qu'aux** {X,[Y],[Z]}
  - . *{the current section, the current chapter, the current article}*

***applies only on** {X,[Y],[Z]}*

There are also the metatextual statements for applying contexts which indicate that some article (section, sentence) applies or not subject to a condition ennunciated in a context.

. {**l'article a, la section s, l'alinéa l**} **s'applique si** contexte1

. ***{article a, section s, sentence l} applies if*** *context1*

. {**l'article a, la section s, l'alinéa l**} **ne s'applique pas** si contexte1

. ***{article a, section s, sentence l} does not apply if*** *context1*

3. THE SACD SUB-SYSTEM GENERATING THE DEONTIC KNOWLEDGE BASE

Starting with the structures obtained after the text compilation (see section 2.1), SACD generates the deontic knowledge base (DKB) . The DKB should have a content which enables an inference engine to process it. We have selected the following principles :

- the DKB is specified with rules (called "deontic rules") in which we can indicate a modality and characterize the object to which each rule is related;
- deontic rules should be independent from each other;
- we can associate one or several rules to each sentence of the initial prescriptive text, and each rule refers to the sentence from which it derives;
- conditions and / or exceptions expressed in a sentence are used to enunciate the basic context of the corresponding deontic rule;
- the basic context of a deontic rule can be enriched if other sentences refer (through internal references) to the sentence from which the rule derives.

In (Moulin and Rousseau 1990-1) we describe the internal structure of deontic rules. In this paper we use a simplified notation for deontic rules such as :

v-rule : **IF** v-context < v-obj, v-deont-op> v-deont-concl

where

v-context corresponds to conditions (and/or exceptions) associated with the rule;
v-obj is the object to which the rule is attached;
v-deont-op is the deontic operator which is associated with the rule;
v-deont-concl is the rule conclusion.

The logical content of the DKB is generated in two steps:

- **Step 1** : for each phrase (v-phrase) found in the prescriptive text, SACD generates a deontic rule (v-rule) with its basic context, without considering internal references.

- **Step 2** : for each metatextual statement, SACD modifies the contexts of the rules associated with the phrases that are in the scope of the corresponding internal reference.

Figure 5 presents the architecture of SACD sub-system which generates the DKB. We use the following conventions : rectangles represent the system functions, rectangles with round corners represent data or knowledge stored in the system; double squares represent the persons which operate the system.

Function P1 uses information which has been generated by SACD parsing system : compiled structures of the text after the microstructure analysis (A1), object type lattice (A2) and the equivalence table for anaphoras (A3). Function P1 does not process sentences with internal references and generates only the deontic rules with basic contexts (A4). In these rules, some anaphoric references are replaced automatically by the corresponding object name. SACD currently replaces personal pronouns which are either subject in the sentence or a complement situated after the verb.

Function P2 generates the deontic knowledge base (A6) on the basis of the deontic rules with basic contexts (A4). Function P2 uses the knowledge structures generated by SACD text analysis sub-system : the compiled version of the text obtained after the analysis of the microstructure (A1), the object type lattice (A2), the equivalence table for anaphoras (A3) as well as a table of the text internal references (A5). Function P2 uses also the compiled version of a rule base used for modifying contexts (A8). These rules compose the core of the generation sub-system and are presented in detail in section 4.2.

Function P3 enables the knowledge engineer (E1) to edit deontic rules (A4) and the DKB (A6) in order to correct some spelling inconsistencies in the propositions that are used by SACD (changing some verb tenses, replacing anaphoras that have not been processed by SACD).

Function P4 enables the knowledge engineer (E1) to edit the rules for modifying contexts (A7) and function P5 transform these rules in a compiled form that is used directly by function P2.

4. GENERATING DEONTIC RULES

4.1 Generating deontic rules with basic context

Function P2 (see figure 5) generates the deontic rules with basic contexts (A4) using the compiled version of the text obtained after the analysis of the microstructure (A1), without considering sentences in wich appear internal references : that's the first step of the DKB generation.

Figure 6 presents the internal form of the deontic rule generated from article 1 (see figure 1). Note that the TABLE-REGLE predicate associates the rule and the corresponding sentence. The deontic rule is specified by the following predicate :

<v-rule, v-phrase, v-article, v-deont-concl, v-truth-value, v-context>

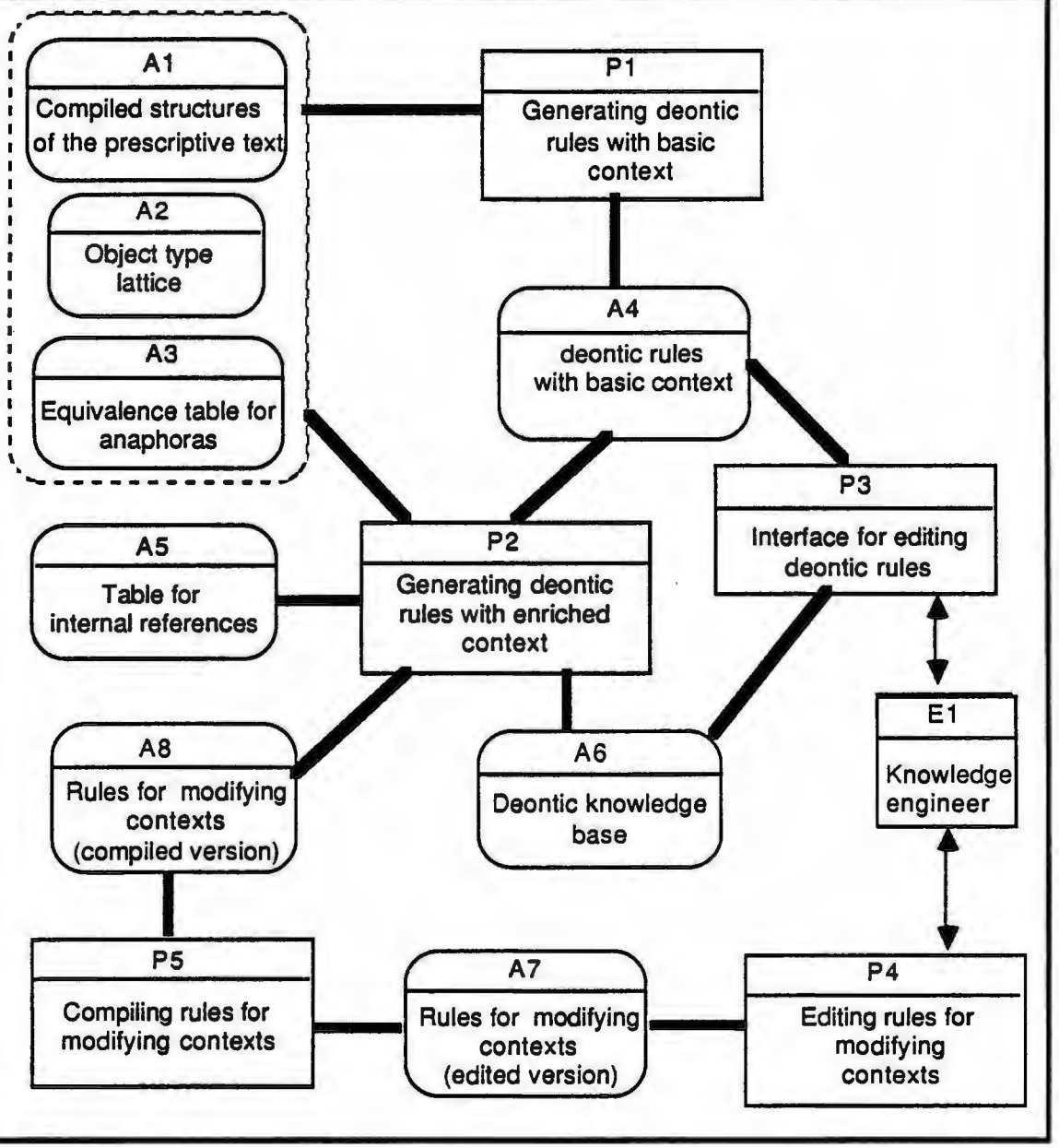
where

v-phrase and v-article correspond respectively to the sentence and the article;

v-deont-concl is the rule conclusion (a predicate in which are embedded the object, the modal

operator and the prescription as well as its truth value); v-truth-value is the truth value associated with the modal operator, v-context corresponds to conditions (and/or exceptions) associated with the rule (for the rule of ficure 6, it is an exception).

Figure 5 **The sub-system generating the deontic knowledge base**



```
TABLE-REGLE(règle1-n1-a1,phr1-n1-a1)  ->;
<objet4-a1-r,"Un".nil,"chauffeur"."de"."taxi".nil,nil>  ->;

<termeV6-a1-r,"refuser"."d"'"."effectuer"."une"."course"."dont"."l"'".
"origine"."est"."située"."à"."l"'"."intérieur"."de"."l"'"."agglomération"."ou"."de".
"la"."région"."pour"."laquelle"."le"."permis"."de"."taxi"."est"."délivré".nil,nil>  ->;

<concl1-a1,objet4-a1-r,<PERM,"peut".nil>,termeV6-a1-r,faux>  ->;

<excp8-a1-r,excp,portée-P,nil,ASS,"la"."destination"."de"."cette"."course".
"soit"."située"."à"."plus"."de".50."kilomètres"."à"."l"'"."extérieur"."des"."limites".
"de"."cette"."agglomération"."ou"."de"."cette"."région".nil,nil,phr1-n1-a1>  ->;

<règle1-n1-a1,phr1-n1-a1,article1,concl1-a1,faux,<excp8-a1-r,faux>>  ->;
```

Figure 6 : Internal code of the deontic rule corresponding to article 1

The following lines show the edited form of the deontic rules, obtained with function P3 (figure 5) : we have in this portion the deontic rules with basic context for articles 1 and 2. The intenal reference introduced by "malgré" (in spite of) in article 2 is not processed at this level.

***** TRAITEMENT DE phr1-n1-a1 *****
REGLE: règle1-n1-a1 PHRASE: phr1-n1-a1

PREMISSE:

SI NON(la destination de cette course soit située à plus de 50 kilomètres à
l ' extérieur des limites de cette agglomération ou de cette région)

CONCLUSION:

NON(Un [chauffeur de taxi] / PERM:peut / refuser d 'effectuer une course
dont l ' origine est située à l ' intérieur de l ' agglomération ou de la
région pour laquelle le permis de taxi est délivré)

****** PROCESSING OF phr1-n1-a1 ******
RULE: règle1-n1-a1 PHRASE: phr1-n1-a1
PREMISE:

IF NOT (the destination of the errand goes beyond the outer limits

of this township or region by more than 50 kilometers)

CONCLUSION:

NOT(A [taxi driver] / PERM: can / decline to run an errand whose origin is situated within the township or region for which the taxi license has been delivered)

***** TRAITEMENT DE phr1-n1-a2 *****

REGLE: règle1-n1-a2 PHRASE: phr1-n1-a2

PREMISSE:

SI NON(un chauffeur de taxi occupe la première place à un poste d ' attente)

CONCLUSION:

---

un [chauffeur de taxi] / OBLS:doit / refuser ses services à un client qui s ' y présente et lui signifier la possibilité de faire sa réquisition au premier taxi en attente

---

****** PROCESSING OF phr1-n1-a2 ******

RULE: règle1-n1-a2 PHRASE: phr1-n1-a2

PREMISE:

IF NOT (a taxi driver holds the first place in a waiting station)

CONCLUSION:

A [taxi driver] / OBL : must / refuse to take the customer who wants to embark, and tell him to ask the first taxi in the row

---

Note that the conclusion of a deontic rule clearly identifies the object (presented between square brackets), the category of the modality (ex. "PERMission") and its exact formulation (ex. "peut" (*"can"*)).

At this stage, the only premise that has been generated for article 1 is the result of the local rule context : the exception which is included in the corresponding sentence : see in figure 1 : "à

moins que ..." (*"unless that"...*). The system will process later the exception introduced by the internal reference "malgré" ("*in spite of ...*") of article 2 (figure 2).

In our model, we process in the same way as conditions, any relative proposition that modifies the object on which holds the prescription : the writer can either express a condition directly or introduce it as a relative proposition. For article 2 (figure 2), "qui n'occupe pas la première place à un poste d'attente" (*"who does not hold the first place in a waiting station"*) is a relative proposition attached to the object "chauffeur de taxi" (*"taxi driver"*). It is transformed as a condition in the premise of the deontic rule associated with article 2 : "SI NON(un chauffeur de taxi occupe la première place à un poste d ' attente)" ("*IF NOT (a taxi driver holds the first place in a waiting station)*").

Note that SACD has replaced the anaphoric reference "qui" (*"who"*) by the name of the referred object "un chauffeur de taxi" (*"a taxi driver"*).

4.2 Rules for modifying contexts (RMC)

The basic context of deontic rules is modified by SACD with function P2 (figure 5). Each time function P2 encounters a new internal reference, it activates the rule for modifying contexts (RMC) which corresponds to the internal reference category. When SACD fires this RMC, it modifies the deontic rules which are associated with the sentences that are referred by the internal reference. The various categories of internal reference presented in section 2.2 have a different impact on the modification of deontic rules. We should mention also that when the scope of an internal reference is a section (or a set of articles) in the original prescriptive text, we will consider that this section (or a set of articles) is associated with a set of phrases (Ph_i pour i =1,n) from which SACD has generated during phase 1 the basic contexts of the following deontic rules :

for i =1,n v-rule$_i$: **IF** v-context$_i$ < v-obj$_i$, v-deont-op$_i$> v-deont-concl$_i$.

As an illustration consider the referential connector "malgré" (*"in spite of"*):

Malgré {l'article a, la section s,} X **op-mod** P_X **si** contexte$_X$

In spite of {article a, section s,} X **log-op$_X$** P_X **if** context$_X$

When processing this internal reference, SACD modifies the context of each rule associated with article a, by adding to each of them NON context$_X$.

For i =1,n v-rule$_i$ becomes :

IF v-context$_i$ **AND NOT** context$_X$ < v-obj$_i$, v-deont-op$_i$> v-deont-concl$_i$

We have defined a language which enables the knowledge engineer to specify and adapt easily the rules for modifying contexts(RMC). Figure 7 presents the specification of the RMC MOD-RULE 11 which is associated with the category of referential connectors like "malgré" (*"in spite of"*). The lines of the rules have been numbered in order to help us to present the comments which follow.

```
MOD- RULE 11 (                                                    ..... I1
 IF <v-ref,in spite of,v-phrase,v-object,nil,l-phrases>           ..... I2
 THEN RULE-TABLE(v-rule,v-phrase)                                 ..... I3
      SEEK(<v-rule,v-phrase,v-article,v-deont-concl,              ..... I4
        v-truth-value-concl,v-context>)
    FOR EACH v-phrase1 OF l-phrases                               ..... I5
    DO(v-context)                                                 ..... I6
        RULE-TABLE(v-rule1,v-phrase1)                             ..... I7
           SEEK(<v-rule1,v-phrase1,v-article1,v-deont-concl1,     ..... I8
        v-truth-value-concl1,v-context1>)
          AFFECT(v-var1,AND(NOT(v-context),v-context1))           ..... I9
         REPLACE(v-context1,v-var1,v-rule1)                       ..... I10
   END-DO                                                         ..... I11
 END-IF);                                                         ..... I12
```

Figure 7 : Rule for modifying contexts associated with "In spite of"

Line I1 corresponds to the identifier of the RMC. Line I2 corresponds to the premise of the RMC : the RMC will be activated by function P2 of the generation sub-system, if it can unify the premise of the RMC with the internal reference which is currently processed. The predicate of line I2 corresponds to the pattern of an internal reference v-ref, introduced by the referential connector "in spite of", associated with phrase v-phrase, attached to object v-object and referring to the list of phrases l-phrases.

The part THEN ... END-IF (lines I3 to I12) corresponds to the body of the RMC and corresponds to the following operations :

- line I3 : SACD identifies the deontic rule associated with the phrase v-phrase;
- line I4 : SACD seeks the elements which compose the deontic rule v-rule (see section 4.1), and specifically the context v-context;
- line I5 : it is an iteration loop on each element v-phrase1 of the list l-phrases which are referred by v-ref;
- line I6 : the body of the iteration loop which has v-context as an argument;
- line I7 : SACD identifies the deontic rule v-rule1 associated with the phrase v-phrase1;
- line I8 : SACD seeks the elements which compose the deontic rule v-rule1 and specifically the context v-context1;
- line I9 : SACD calculates the new context by affecting to the intermediate variable v-var1 the result of the logical operation
  (v-context) AND NOT (v-context1);

- line l10 : SACD replaces v-context1 by its new value from v-var1;
- line l11 : end of the iteration loop.

Each internal reference category is associated with a RMC specified in a similar way. These rules are compiled by function P5 of the generation sub-system (figure 5) in Prolog code (A8 in figure 5) which is processed bu function P2 (figure 5).

4.3 An example where a RMC is used

We present here the trace of the program which enriches the context of deontic rules (function P2 figure 5) when it is applied on the case of the internal reference of article 2 (figure 2). The program activates the rule for modifying contexts MOD-RULE 11 which has been described in the preceding section.

SACD starts by deleting the rule which was generated from article 1 during the generation of deontic rules with basic context (see section 4.2). Then the RMC creates the new deontic rule associated with article 1, after applying the internal reference "malgré" (*in spite of*) in article 2. Note that the context of the deontic rule has been enriched by

"SI un chauffeur de taxi occupe la première place à un poste d ' attente",

(*"IF a taxi driver holds the first place in a waiting station"*)

which is the negation of the context of the rule which is associated with article 2 (see section 4.2)

REFERENCE: réf-C2-a2 PHRASE: phr1-n1-a2

-> REGLE SUPPRIMEE: ..... ***the following rule is deleted***

REGLE: règle1-n1-a1 PHRASE: phr1-n1-a1

PREMISSE:

SI NON(la destination de cette course soit située à plus de 50 kilomètres à
l ' extérieur des limites de cette agglomération ou de cette région)

CONCLUSION:

NON(Un [chauffeur de taxi] / PERM:peut / refuser d 'effectuer une course
dont l ' origine est située à l ' intérieur de l ' agglomération ou de la
région pour laquelle le permis de taxi est délivré)

-> REGLE AJOUTEE: ..... ***the following rule is added***

REGLE: règle1-n1-a1 PHRASE: phr1-n1-a1

PREMISSES:

SI un chauffeur de taxi occupe la première place à un poste d ' attente
ET SI NON(la destination de cette course soit située à plus de 50 kilomètres à l ' extérieur des limites de cette agglomération ou de cette région)

CONCLUSION:

NON(Un [chauffeur de taxi] / PERM:peut / refuser d ' effectuer une course dont l ' origine est située à l ' intérieur de l ' agglomération ou de la région pour laquelle le permis de taxi est délivré)

RULE: règle1-n1-a1 PHRASE: phr1-n1-a1

PREMISES:

IF a taxi driver holds the first place in a waiting station
AND IF NOT(the destination of the errand goes beyond the outer limits of this township or region by more than 50 kilometers)

CONCLUSION:

NOT(A [taxi driver] / PERM: can / decline to run an errand whose origin is situated within the township or region for which the taxi license has been delivered)

SACD has been used to analyse a variety of regulation texts from the Governments of Quebec and Canada written in French : Réglement de transport par taxi, réglement sur le traitement des déchets solides, loi sur l'accès à l'information etc. We have also analysed some portions of the National Building Code. Deontic knowledge bases have been genereated by SACD from these analyses.

One immediate use of these knowledge bases consists in checking the influence of the numerous cross references found in the original text. In several cases we have found some rules which are logically too complex to be understood well by the regulation text users. That happened for instance with the analysis of a section of the French version of the National Building Code (NBC) in which there are many cross-references (section on firewalls). When we examined the knowledge base generated by SACD, we found 3 rules that were so complicated that we could not understand how NBC users could apply them in practice. When we showed these results to NBC specialists, they confirmed the validity of the rules and agreed with us that the current NBC version can give some trouble to its users.

The knowledge bases generated by SACD can also be used for building consultation systems which will check cases against the regulation provisions. We are currently working in this direction.

5. CONCLUSION

In this paper we have presented a model which enables us to analyse the logical content of prescriptive texts. We have focused on the detection and processing of internal references which provide most complexities that people face when they try to understand prescriptive texts. This model is the theoritical basis of SACD, a knowledge acquisition system which analyses the logical content of prescriptive texts and generates "deontic" knowledge bases with a logical content equivalent to the original text content. We have described in more details SACD sub-system which generates the deontic knowledge base.

The system is implemented in Prolog and runs on Macintosh workstations. It has been tested on a corpus of texts of regulations from the Government of Quebec. Various extensions are under study for the short and mid terms : the use of the deontic knowledge base to check the coherence and complexity of prescriptive text logical content; the integration of SACD functionalities in a system which will support writers when they elaborate and formulate prescriptive texts.

Acknowledgments

This research is supported by CEFRIO, le Centre francophone de recherche en informatisation des organisations (grant 1987-90), and by the Natural Sciences and Engineering Research Council of Canada.

NOTE

1. We use the terms "deontic knowledge base" as a reference to the deontic logic which is used as a basis for the analysis of the logical content of regulation texts.

REFERENCES

ACM (1987) Proceedings of the First International Conference on Artificial Intelligence and Law, Boston MA, May 1987, ACM Press.

ACM (1989) Proceedings of the Second International Conference on Artificial Intelligence and

Law, Vancouver BC, June 1989, ACM Press.
BENCH-CAPON T. J. M. (1987), G. O. Robinson, T. W. Routen, M. J. Sergot, Logic programming for large scale applications in law, in (ACM 1987).
BOOSE J. H. (1990), B. R. Gaines editors, proceedings of the Fifth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, November 1990.
BOY G. (1990), Acquiring and refining indices according to context, in (Boose, Gaines 1990).
GOUVERNEMENT DU QUEBEC (1987), Règlement sur le transport par taxi, révisé 1987.
GARDIES J-L (1980), La structure logique de la loi, *Archives de philosophie du droit*, tome 25, pp 109-121.
JACOBS P. (1988), ZERNIK U., Acquiring lexical knowledge from text : a case study, Proceedings of the AAAI88, St. Paul, Min.
KALINOWSKI G. (1972), *La logique des normes*, Presses Universitaires de France.
LAGACE J. (1987), Expressions utilisées dans les renvois, dans *Légistique, bulletin de rédaction législative et réglementaire*, Société québécoise d'information juridique, n1 janvier 1987, pp 21-30.
LEBOWITZ M. (1988), The use of memory in text processing, in *Communications of the ACM*, vol 31, n 12 pp 1483-1502.
MOULIN B. (1988), Réflexions sur la conception et l'utilisation de bases de connaissances déontiques, Rapport de recherche CEFRIO.
MOULIN B. (1989), D. Rousseau, Conception de bases de connaissances déontiques pour la consultation de textes de réglement, proceedings of the Eighth Workshop on Expert Systems and their Applications, Avignon France May 1989.
MOULIN B. (1990-1), D. Rousseau, Structuration d'une base de connaissances à partir du contenu de textes prescriptifs, Conférence sur le traitement du langage naturel, in proceedings of the Tenth Workshop on Expert Systems and their Applications, Avignon France May 1990.
MOULIN B. (1990-2), D. Rousseau, Designing deontic knowledge bases from regulation texts, in the Journal *Knowledged-Based Systems*, vol 3 n2, pp 108-120 , June 1990.
MOULIN B. (1990-3), D. Rousseau, A knowledge acquisition system for analysing prescriptive texts, in (Boose, Gaines 1990).
REIMER U. (1990), Automatic knowledge acquisition from texts : learning terminological knowledge via text understanding and inductive generalization, in (Boose, Gaines 1990).
TREMBLAY R. (1987), Le renvoi, dans *Légistique, bulletin de rédaction législative et réglementaire*, Société québécoise d'information juridique, n1 janvier 1987, pp 3-20.
TREMBLAY R. (1984), R. Journeault, J. Lagacé, *Guide de rédaction législative*, publication du Gouvernement du Québec, Société québécoise d'information juridique.
VELARDI P. (1989), Pazienza M. T., Magrini S., Acquisition of semantic patterns from natural corpus of texts, in SIGART Newsletter, Special issue on Knowledge Acquisition, n. 108.
ZERNIK U. (1989), Lexicon acquisition : Learning from corpus by capitalizing on lexical categories, in proceedings of IJCAI89, Detroit, Mich.

Acquisition of Knowledge for Sharing and Reuse

Robert Neches
USC / Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
213 / 822-1511
Neches@ISI.edu

ABSTRACT

Building new knowledge-based systems today usually entails constructing new knowledge bases from scratch. It could instead be done by assembling together reusable components, requiring system developers only to worry about creating the specialized knowledge and reasoners new to the specific task of their system. Their new system would inter-operate with existing systems, using them to perform some of its reasoning. In this way, declarative knowledge, problem solving techniques and reasoning services could all be shared among systems. This would facilitate building bigger and better systems cheaply. The infrastructure to support such sharing and reuse would lead to greater ubiquity of these systems, potentially transforming the knowledge industry. This article presents a vision of the future in which knowledge-based system development and operation is facilitated by infrastructure and technology for knowledge sharing. It describes an initiative currently underway to develop these ideas, and suggests steps that must be taken in the future to try to realize this vision.

OVERVIEW

Ever since the mid-seventies, researchers have recognized that capturing knowledge is the key to building large and powerful AI systems. In the intervening years, we have also found that representing knowledge is difficult and time-consuming. Although we have developed tools to help with knowledge acquisition, knowledge base construction remains one of the major costs in building an AI system because, for almost every system we build, a new knowledge base must be constructed from scratch. As a result, most systems remain small to medium size. Even if we build several systems within a general area such as medicine or electronics diagnosis, significant portions of the domain must be re-represented for every system we create. The cost of this duplication of effort has been high, and will become prohibitive in the future as we attempt to build larger and larger systems. To overcome this barrier and advance the state of the art, we must find ways of preserving existing knowledge bases and of sharing, reusing and building upon them.

A Vision: Knowledge Sharing

In this article, we present a vision of the future in which knowledge sharing is commonplace. If this vision is realized, building a new system will rarely involve constructing a new knowledge base from scratch. Instead, the process of building a knowledge-based system will start by assembling together reusable components. Portions of existing knowledge bases would be re-used in constructing the new system, and special-purpose reasoners embodying problem solving methods would similarly be brought in. Some effort would go into connecting these pieces together, creating a "custom shell" with pre-loaded knowledge. However, most system development effort could become focused on creating only the specialized knowledge and reasoners that are new to the specific task of the system under construc-

tion. In our vision, the new system could inter-operate with existing systems and pose queries to them to perform some of its reasoning. Furthermore, extensions to existing knowledge bases could be added back to shared repositories, expanding and enriching them.

Over time, large rich knowledge bases, analogous to today's databases, will evolve. In this way, both declarative knowledge, problem solving techniques and reasoning services could be shared among systems. The cost to produce a system would decrease. To the extent that well-tested parts were reused, a system's robustness would increase.

This vision alters the face of information systems for end-users in three ways. First, it will provide sources of information that serve the functions of books and libraries, but are more flexible, more easily updated, and easier to query. Second, it will enable the construction and marketing of pre-packaged knowledge services, allowing users to invoke (rent or buy) services. Third, it will make it possible for end-users to tailor large systems to their needs, by assembling knowledge bases and services rather than programming them from scratch.

We also expect changes and enhancements in the ways that developers view and manipulate knowledge-based systems. In particular, we envision three mechanisms that would increase their productivity by promoting sharing and reuse of accumulated knowledge. First among these are libraries of multiple layers of reusable knowledge bases which could be either incorporated into software or remotely consulted at execution time. At a level generic to a class of applications, layers in such knowledge bases capture conceptualizations, tasks, and problem solving methods. Second, system construction will be facilitated by availability of common knowledge representation systems and means for translation between them. Finally, this new reuse-oriented approach will offer tools and methodologies allowing developers to find and utilize library entries useful to their needs, as well as pre-existing services built on those libraries. These tools will be complemented by tools allowing developers to offer back their work for inclusion in the libraries.

The Knowledge Sharing Effort

We are not technically ready to realize this vision at present. Instead, we must work toward it incrementally. For instance, there is no consensus today on the appropriate form or contents of the shared ontologies that we envision. In order for that consensus to emerge, we need to engage in exercises in building shared knowledge bases, then extract generalizations from the set of systems that emerge and capture those generalizations in a standard format that can be interpreted by all involved. This requires the development of some agreed-upon formalisms and conventions, at the level of an interchange format between languages or a common knowledge representation language.

Simply enabling the ability to share knowledge is not enough for the technology to have full impact, however. Development and use of shared ontologies cannot become cost-effective unless the systems utilizing them are highly interoperable with both AI and conventional software, so that large numbers of systems can be built. This means that software interfaces to knowledge representation systems are a crucial issue.

The Knowledge Sharing Effort, sponsored by AFOSR, DARPA, the Corporation for National Research Initiatives, and NSF, is an initiative to develop the technical infrastructure to support sharing of knowledge among systems. The effort is organized into four working groups, each of which is described briefly below, and in greater detail in the remainder of the article.

The **Interlingua** group is developing an approach to translating between knowledge representation languages. Their approach involves developing an intermediary language, a *knowledge interchange format* or *interlingua*, and a set of translators to map into and out of it from existing knowledge representation languages. To map a knowledge base from one

representation language into another, a system builder would use one translator to map the knowledge base into the interchange format, and then use another translator to map from the interchange format back out to the second language.

The **Knowledge Representation Systems Specification** (KRSS) working group is taking another, complementary tack toward promoting knowledge sharing. Rather than translating between knowledge representation languages, the KRSS group is seeking to promote sharing by removing arbitrary differences among knowledge representation languages within the same paradigm. This group is currently working on a specification for a knowledge representation system that brings together the best features of languages developed within the KL-One paradigm. Similar efforts for other families of languages are expected to follow.

The **External Interfaces** working group is investigating yet another facet of knowledge sharing. They are developing a set of protocols for interaction that would allow a knowledge-based system to obtain knowledge from another knowledge-based system (or possibly a conventional database) by posting a query to that system and receiving back a response. The concerns of this group are to develop the protocols and conventions through which such an interaction could take place.

Finally, the **Shared, Reusable Knowledge Bases** group is working on overcoming the barriers to sharing arising from lack of consensus across knowledge bases on vocabulary and semantic interpretations in domain models. The ontology of a system consists of its vocabulary and a set of constraints on the way terms can be combined in order to model a domain. All knowledge systems are based on an ontology, whether implicit or explicit. A larger knowledge system can be composed from two smaller ones only if their ontologies are consistent. This group is trying to ameliorate that problem by fostering the evolution of common, shareable ontologies. A number of candidate reusable ontologies are expected to fall out of the work. However, the ultimate contribution of the group lies in developing an understanding of the group processes that evolve such products, and the tools and infrastructure needed to facilitate creation, dissemination, and reuse of domain-oriented ontologies.

ARCHITECTURES OF THE FUTURE

In this section, we will elaborate our vision by describing what we hope to enable concerning both knowledge bases and the systems which utilize them. In doing so, we will look at both how they are structured and the process by which they will be built.

Structural and Development Models for Knowledge Bases

John McDermott, in a recent AAAI-90 panel on software engineering, described how AI could make software development easier: write programs which act as a framework for handling any instance of a class of problems that matches its abstractions (McDermott 1990).

Knowledge-based systems can provide such frameworks in the form of top-level declarative abstraction hierarchies, which an application builder elaborates to create a specific system. Essentially, hierarchies built for that purpose represent a commitment to provide specific services to applications that are willing to adopt their model of the world.

When these top-level abstraction hierarchies are represented with enough information to lay down the ground rules for modeling a domain, we call them *ontologies*. An ontology defines the basic terms and relations comprising the vocabulary of a topic area, as well as the rules for combining terms and relations in order to define extensions to the vocabulary. An example is the MKS generic model of manufacturing steps (Pan, Tenenbaum, and Glicksman 1989), illustrated in Figure 1 along with a set of application-specific extensions for semiconductor manufacturing. The frame hierarchy in MKS defines classes of concepts that the

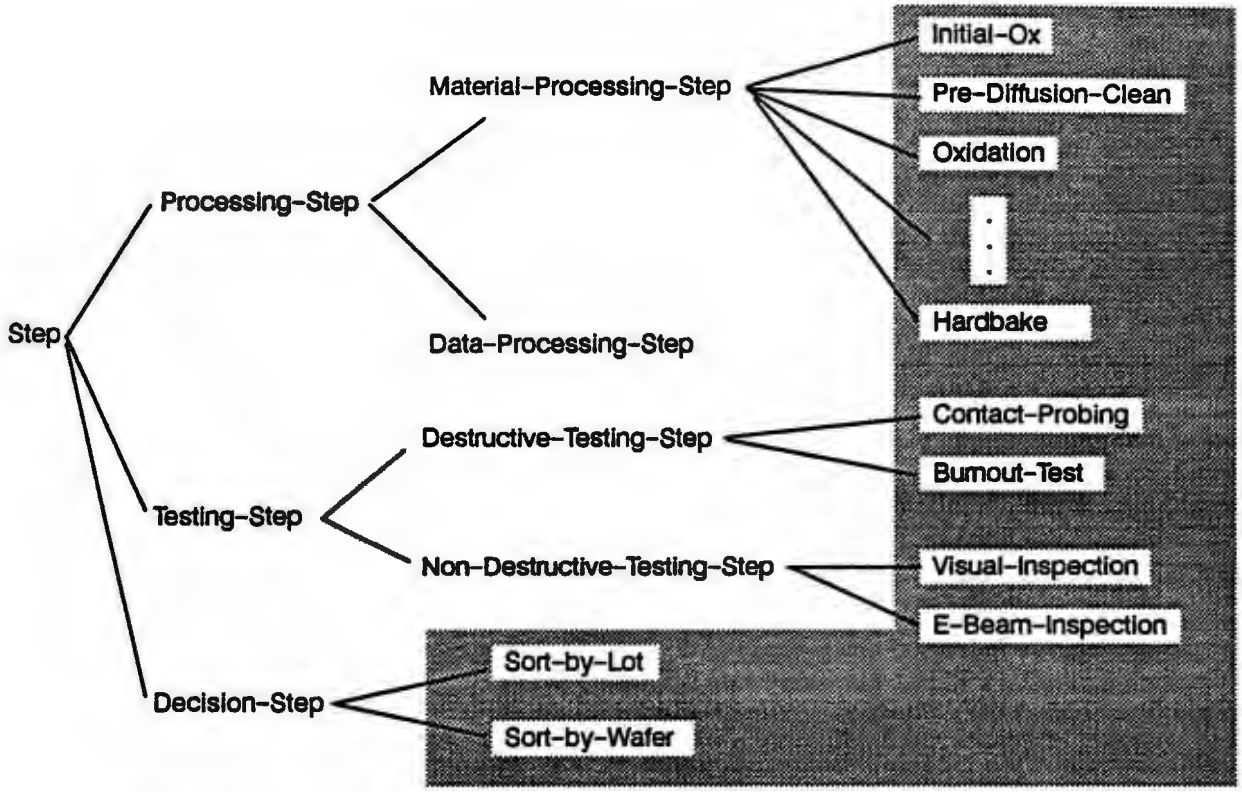


Figure 1: The MKS ontology of manufacturing operations, elaborated with knowledge specific to semiconductor manufacturing

system's reasoning modules (e.g., schedulers and diagnosers) are prepared to operate upon. The slots and slot restrictions on those frames define how one must model a particular manufacturing domain in order to enable use of those modules.

The MKS example is hardly unique; a number of systems have been built in a similar spirit. The notion that generic structure can be exploited in building specialized systems has been argued for a long time by Chandrasekaran (1983, 1986) and more recently by Steels 1990. The notion has also long been exploited in knowledge acquisition work, e.g., in systems like MORE (Kahn, Nowlan, and McDermott 1984) and ROGET (Bennett 1984). However, although every knowledge-based system implicitly or explicitly embodies an ontology, currently, ontologies are rarely shared across systems. A lot of effort may have been duplicated in creating separate models — for example, Stanford's Summer Ontology Project found that each of more than half a dozen collaborators had a knowledge-based system that explicitly or implicitly modeled mechanical devices in comparable terms referring to modules, ports and interconnects (Gruber 1991). Even minor differences in terminology hamper effective communication or knowledge sharing. Thus, a library of shared ontologies would facilitate building systems by reducing the effort invested in reconciliation and reinvention.

Structural and Development Models for Knowledge-Based Systems

Figure 2 illustrates the envisioned organization of an individual knowledge-based application system. The local system consists of a set of services that operate upon combined (or indistinguishable) knowledge- and data-bases. One uniform user interface management system mediates interactions with humans, and a set of agents mediates interaction with other systems.

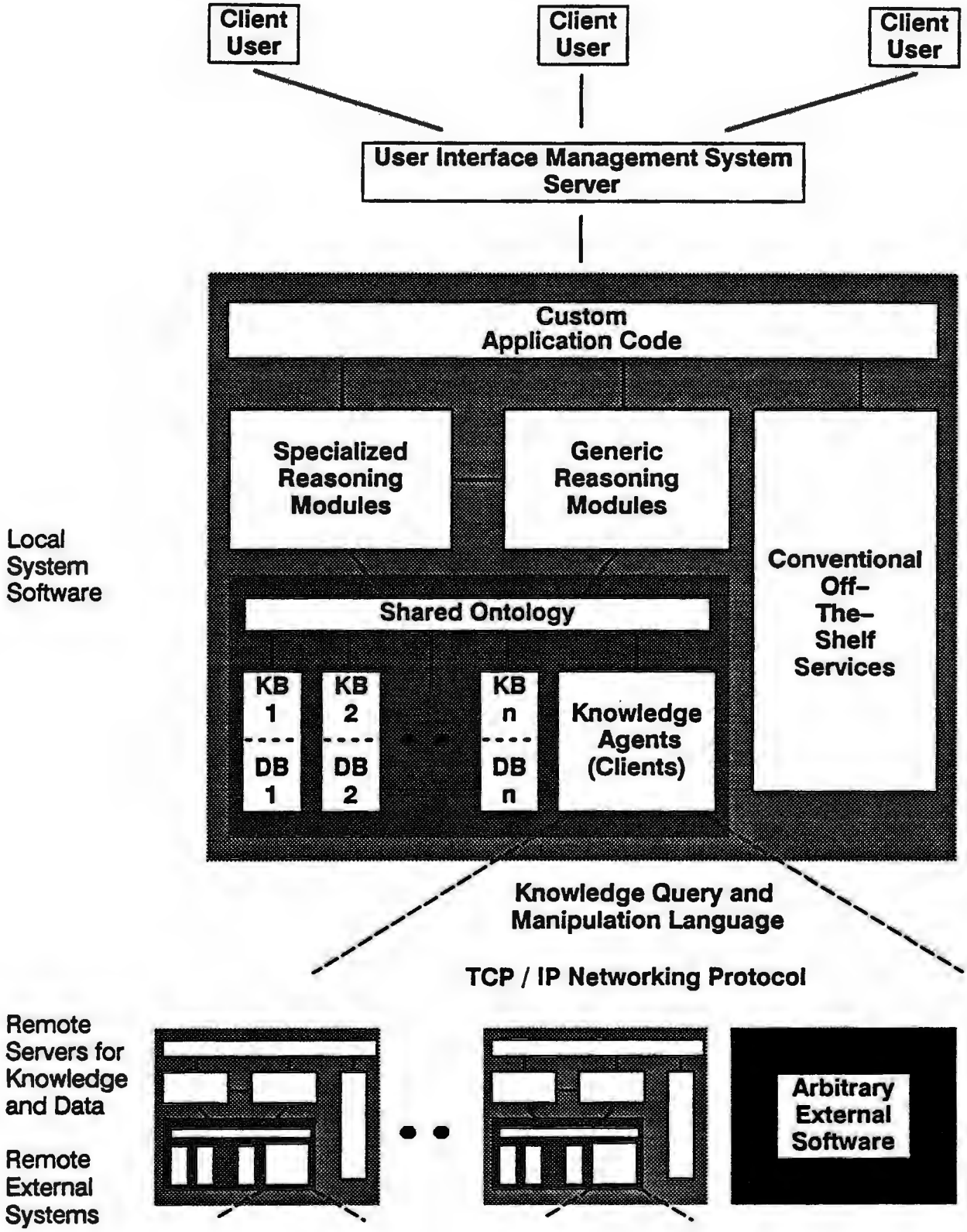


Figure 2: Architecture of a knowledge-based system

The services comprising the application consist of various reasoning modules, which can be defined either at the level of generic reasoning and inference mechanisms (e.g., forward chaining, backward chaining, unification, etc.) or at the task level (e.g., planners, diagnosers, etc.). These modules would typically be acquired off the shelf. At either level, however, they may be augmented by special–purpose reasoning modules that provide custom capabilities for exploiting particular characteristics of the application domain. In addition to these reasoning modules, the application's services are also likely to include a number of modules providing conventional capabilities, such as spreadsheets, electronic mail, hypermedia, calendar systems, statistical packages, and accounting systems.

To perform the tasks of the overall system, modules will need to interact internally (i.e., knowledge base to knowledge base), and externally (i.e., knowledge/data base to other knowledge–based systems or to arbitrary outside software applications). For the latter, they will need a language for encoding their communications. SQL serves that function for conventional database interactions and appears likely to continue to be used in the future; we will call the analogous programmatic interface for knowledge–based applications *KQML*, for *Knowledge Query and Manipulation Language*. KQML will consist of a language for specifying wrappers that define messages communicated between modules. These wrappers will surround declarations that will express whatever knowledge the sending module must pass to the receiving module.

The uniform KQML communication protocol will facilitate modular construction of system components and thus facilitate the development of libraries of reusable components. Such libraries will contain generic reasoners such as truth maintenance systems, task–specific reasoners (planners, diagnosers, etc.), and domain–specific reasoners (e.g., electrical circuit simulators). The libraries will also contain a number of ontologies covering both structural knowledge and problem–solving knowledge. Some will be domain oriented, and some will correspond to particular reasoners. Also residing in those libraries will be a collection of knowledge representation system implementations to choose among. We expect that there might be several different representational paradigms, or families, available in the library. For each family, we expect there to be multiple implementations of a common core language.

As Figure 3 illustrates, application–specific systems will be developed by assembling components from a library into a customized shell, which is then used to develop the application. The first task of system developers would be to configure that specialized shell. This will involve selecting ontologies, specialized reasoning modules, and a knowledge representation system from the library. As in any configuration task, there are likely to be constraints that must be respected. For example, a particular specialized scheduling module might assume that tasks to be scheduled are modeled according to a particular ontology. The entries in the library must provide enough information about themselves to allow system developers to understand those constraints.

Once the ontologies, specialized reasoning modules and a knowledge representation system have been selected and assembled, the system developers have a specialized shell for their application. This shell differs from today's shells in that it will come with built–in knowledge, not just specialized programming features. The system developers' tasks are then to flesh out the knowledge base, add whatever custom application code is necessary, and write software interfaces to any other systems the new application will work with.

In configuring this specialized shell, it would be highly desirable if there were support for translation between representations. Then, developers would not be obliged to use the original implementation if their application had very different performance or functionality requirements.

Note that the approach we have been describing scales with the evolution of infrastructure for knowledge sharing. It is feasible with existing technology. If and when common knowledge

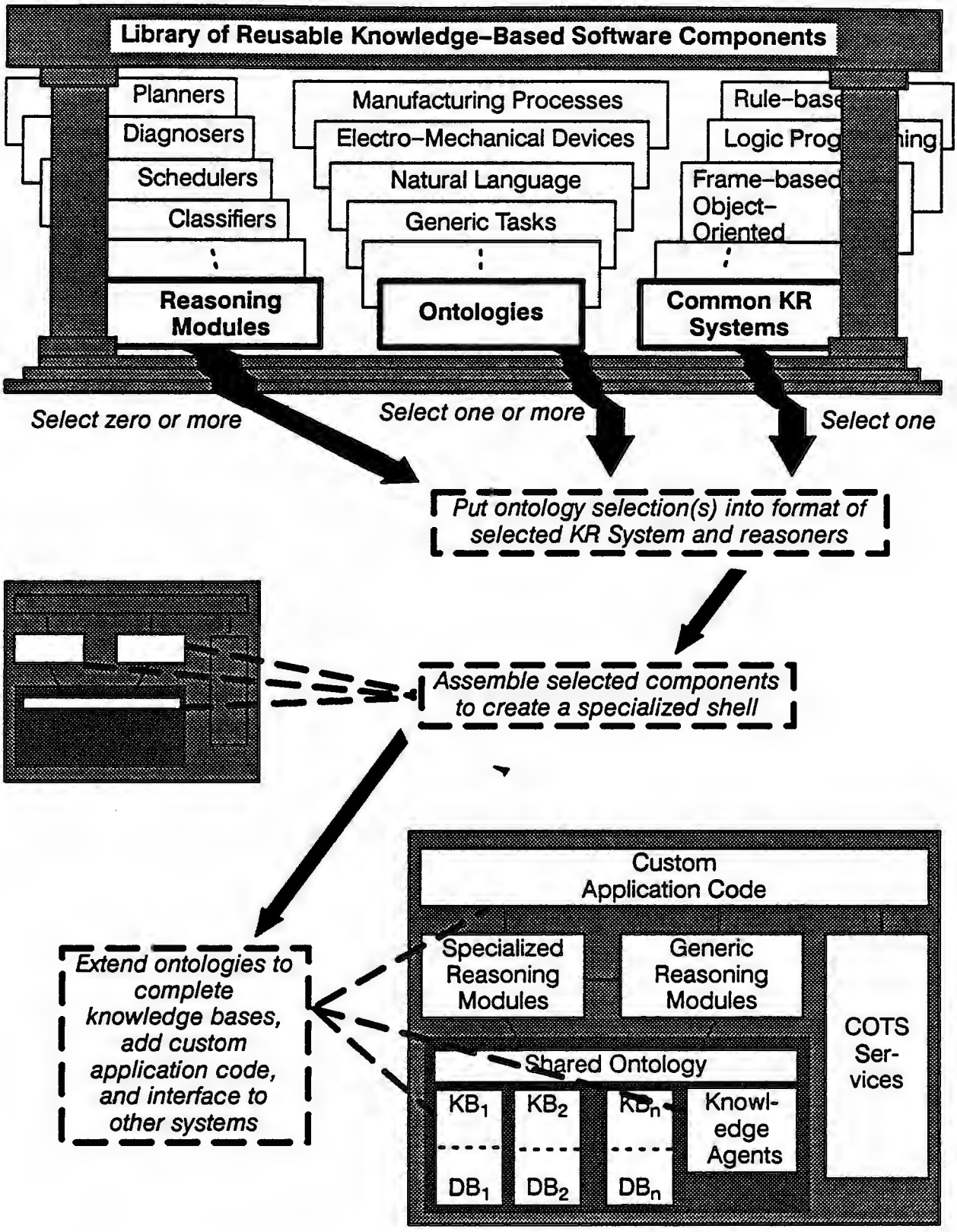


Figure 3: Envisioned phases in defining a knowledge-based system

representation systems become available, their use would broaden the portability and reusability of a given library. Similarly, development and dissemination of techniques for translation between different representation languages would also broaden the usage of a library.

Figure 4 contrasts our model of the lifecycle for knowledge-based software to that followed in expert system software today; the notion of libraries is a key difference. In today's models of expert system software development, exemplified by Waterman's book on expert systems (Waterman 1986), there are at least four classes of participants in the lifecycle. *Tool builders* create shells and development environments. *System engineers* take those tools and, by interviewing *domain experts*, create an initial version of a knowledge-based system. Then, together with the domain experts, the system engineers test, extend, and refine the system. Finally, *end users* put the system to use. The vision we are proposing changes the nature of the system engineer's work, and has the potential to merge roles by giving domain experts more ability to act as system engineers. It also adds a new participant to the infrastructure: a *knowledge librarian*, who serves as a keeper of reusable ontologies and implementations of specialized reasoning methods.

The knowledge librarian works with tool builders on tools and development environments to help system engineers browse and select modules from the library. System engineers import, extend, and customize these modules. They may retain their customizations in private

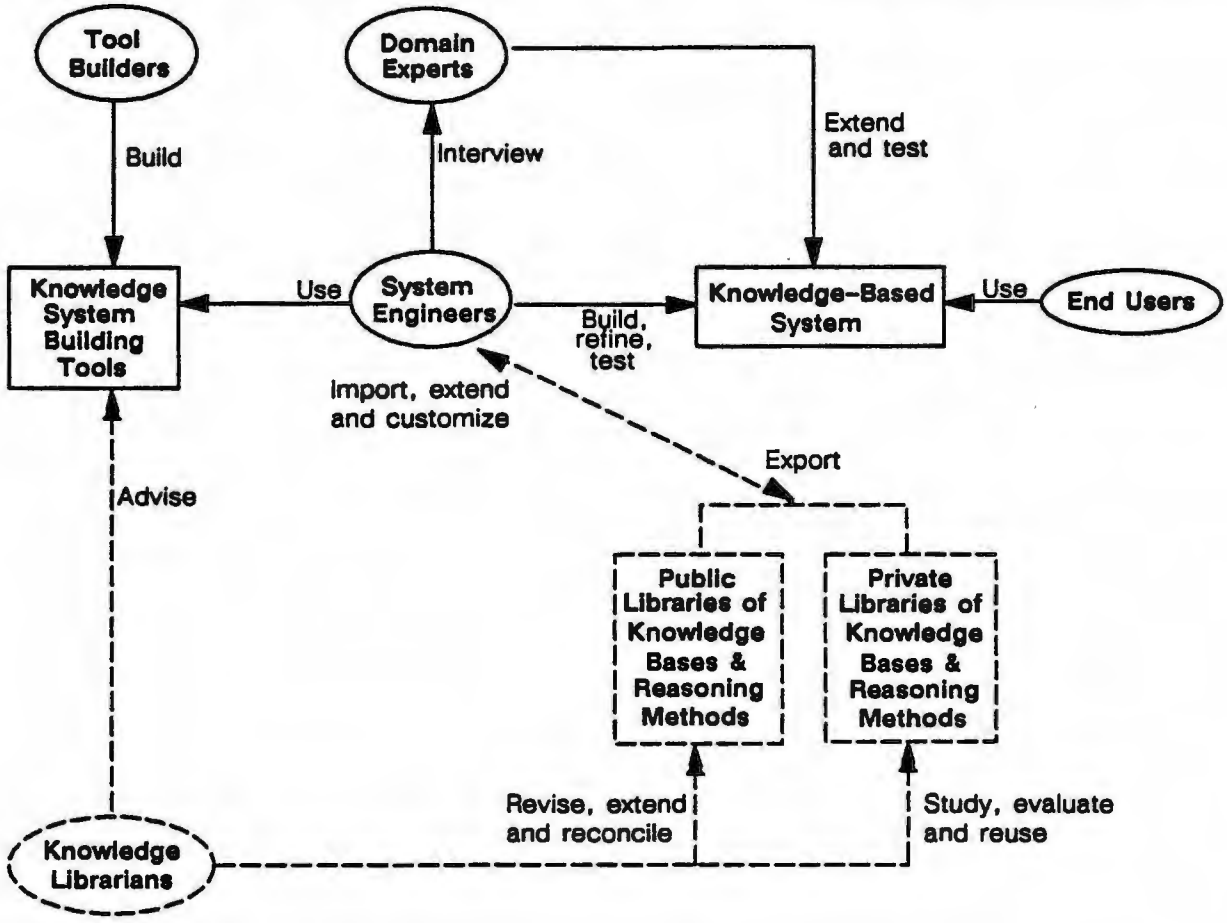


Figure 4: Current versus envisioned models of the AI software lifecycle

libraries for sharing and reuse within a sub–community. Alternatively, they may offer their developments back to the library for possible inclusion as extensions to the existing set of knowledge.

One of the crucial functions which must be performed in the management of a knowledge library is the determination of when submitted knowledge is ready for release. If knowledge is simply added and edited by all participants without some discipline, then it will be difficult to achieve the level of reliability and stability needed for practical software development.

Adding a knowledge library is a significant change to knowledge base methodology. It transforms the system–building process away from using power tools for constructing essentially custom, one–of–a–kind systems. Instead, system–building becomes a process of selecting and combining a range of components. The system engineer becomes much more like a builder of homes, and much less like a sculptor. A home builder has a range of components and materials, from bricks and two–by–fours to prefabricated walls or even rooms. A home builder has a choice of building each new home from small components that give a lot of design flexibility or larger components that trade–off reduced design options for reduced construction effort. Similarly, system engineers would find a range of grain sizes among objects in the knowledge library, and would be empowered to make analogous choices.

Tools for the Evolution of Ontologies and Knowledge Libraries

This section outlines one approach to supporting the development of large shared knowledge bases. This approach is being taken in one of our projects at ISI which is collaborating with the Knowledge Sharing Effort. The approach, embodied in a system called SHELTER, provides an integrated solution to two companion problems: managing the size and complexity of the knowledge base, and managing the size and complexity of the activities required to correctly use, modify, or extend such knowledge bases. SHELTER facilitates building ontologies of the kind that will be needed in these libraries of the future.

The vision of future AI computing given above implies three requirements on the nature of a useful knowledge base development environment:

1. The need for heavy support of browsing, retrieval, and merging functions. Any software object, and knowledge bases are no exception, rapidly becomes unmanageable if there is overlap, duplication, or inconsistency within the system. Thus, SHELTER should help ensure that developers never create when they could reuse. For that to be the case, it follows necessarily that it must be very easy for developers to inspect the knowledge base. After all, it is very difficult for developers to reuse or adapt existing specifications if they do not know that those specifications exist. With large knowledge bases, this implies that we cannot rely on people scanning files of text or looking at graphical displays of lattices. There is simply too much information for that to be feasible. Instead, we must provide winnowing devices, that first find a small set of candidates for inspection by these traditional means.

2. The need to support collaboration and communication between multiple developers. Large systems become too big to be developed, much less maintained, by single individuals. Over time, responsibility for development and maintenance changes hands. Furthermore, if knowledge bases are to be reused, the knowledge in them must be represented in a fashion that simultaneously meets the needs of various systems with very different underlying expertise. It is unlikely that one individual can understand all the relevant issues. (For example, these might range from design vs. manufacturing concerns when we try to describe an electro–mechanical system, to temporal–modeling vs. natural language concerns when we try to generate text describing a simulation.) Capturing metaknowledge ameliorates this problem but does not make it go away; it may be too expensive to capture in advance everything one might need, or previous constraints may no longer apply. Thus, an acceptable representa-

tional solution may only be obtainable by *negotiation* between representatives of different applications or of different areas of expertise. This must be supported.

3. *The need for large amounts of metaknowledge when handling large amounts of knowledge.* As knowledge bases grow large, it becomes very difficult for even the original developers of the abstractions captured in a knowledge base to keep track of the assumptions that were made about how that knowledge is to be used. This problem is compounded when the knowledge bases are intended for sharing and reuse, and it is not even the original developers who are working with the constructs in the knowledge base.

The starting points for SHELTER are two modules: the BACKBORD browser and the TINT note–taking aid.

Browsing and Retrieval. BACKBORD is our first implementation of a specification by reformulation shell — a paradigm of human/computer interaction in which the user develops, by successive approximations, a description of the objects a system is to manipulate and/or the behavior it is to evince (Yen, Neches, DeBellis, Szekely, Aberg 1991). At each step along the way, the system uses the current form of the description to generate examples from the knowledge base for the purpose of showing the user what kinds of alternatives exist for further refining the description. Although users can always type in what is wanted, the system also lets them indicate any field displayed on the screen and gives them ways to state how to use the information in that field to modify the description.

Although currently only for a few of the constructs needed, BACKBORD provides an encapsulation for three sets of capabilities that a knowledge base developer needs for manipulating entities. These capabilities are: finding, viewing, and editing. For the purpose of *finding* representation entities, descriptions can be treated as retrieval specifications and used to extract entities from the knowledge base which satisfy the description. For the purpose of *viewing* representational entities, descriptions can be treated as display specifications and manipulated to define alternative displays. For the purpose of *editing*, descriptions can be treated as specifications to be installed in the knowledge base (either installed anew or as a replacement of the entity from which they were originally derived).

Thus, BACKBORD provides a uniform interactive interface to the key functionality of the development environment. This is significant because we believe that BACKBORD's specification by reformulation interaction paradigm is particularly well–suited to helping developers access and reuse existing pieces of specification whenever possible.

For example, Figure 5 illustrates a sample specialized find/view/edit display for inspecting LOOM knowledge base instances . The current form of the description is displayed in the upper left of the three windows shown. At this point in time, the user is looking at portions of a knowledge base which represents presentation and control knowledge in an intelligent user interface. The description corresponds to instances of the class of things which are a **Command** and have at least four **Command Inputs**. The upper right window in the figure is displaying a list of instances satisfying that description, which BACKBORD has retrieved to generate examples. From that list, the user has selected one, **GENERIC–PASTE–FEATURE**, and asked BACKBORD to display a fuller view of it, which appears as the bottom of the three windows. If something in the example suggested to the user a way to refine the description, the user can select the appropriate field and use it to refine the description. The way that it is used takes advantage of the knowledge base, and goes beyond mere copying. For instance, one of the attribute fields in the displayed example is **Execute Method**, which has the value, **EXECUTE–WHEN–READY**. The user might copy that attribute–value pair into the description, but also might decide instead to use some other value which can be derived from that one, e.g., generalizing it to any command whose **Execute Method** is a member of some superclass of **EXECUTE–WHEN–READY** rather than using the attribute value itself.

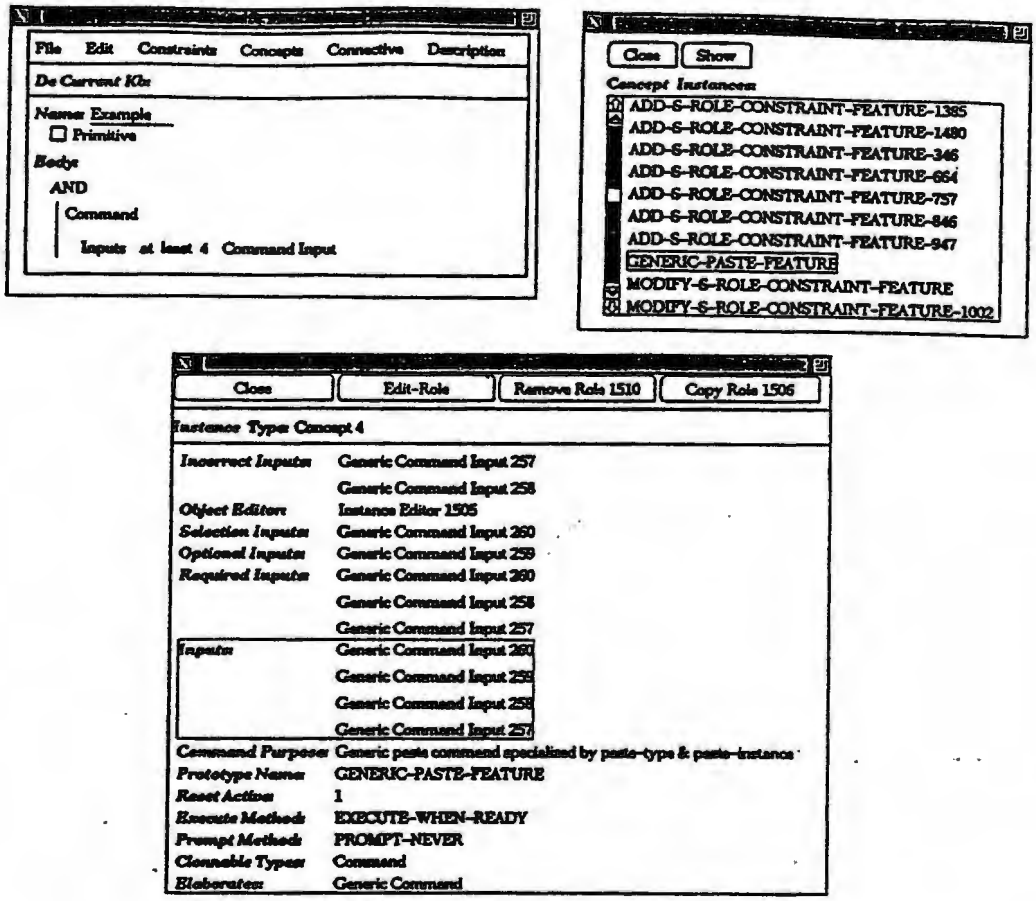


Figure 5 A View/Find/Edit Display for LOOM Knowledge Base Instances

Collaboration and Communication among Multiple Designers. One of the big virtues of this style of interaction is that the developer is exposed to the vocabulary defined in the knowledge base, not just the specific objects already defined using that vocabulary. This maximizes the opportunity to reuse existing terminology when a developer creates new entities. This is useful for individual designers working with knowledge bases too big to easily grasp, and is essential for designers working in teams, who might otherwise have never even known of vocabulary developed and utilized by others. This is critical to ensuring that developers share representational structures, which is crucial to understandability and maintainability of large knowledge bases.

SHELTER's find/view/edit technology builds on BACKBORD, by customizing it for the different kinds of representational constructs in the knowledge representation language. In addition, since BACKBORD is extensible, the specification the find/view/edit technology can be extended and customized for application-specific objects, thus supporting the creation of specialized knowledge-base development environments.

TINT (The Intelligent Note-Taker) acts as a user interface which allows users to record information in the form of "semi-structured" notes that are attached to concepts in a knowl-

edge base (Harp and Neches 1988). Both user and system can create and retrieve notes, and use them to communicate with each other and to store information for their own use. We refer to notes as *semistructured* because the specification method for their internal structure defines attributes (fields) for each note type, and may — or may not — require that all values of those attributes be represented in machine–interpretable form. Thus, the computer can always operate with at least partial understanding, because the notes have some structure to them and are associated with knowledge base entries that the computer can be programmed to understand. We represent notes in a frame–based type hierarchy like Malone's semistructured objects (Lai and Malone 1988). In Malone's scheme, however, there is no special provision for retrieval. In TINT, notes are associated with concepts in a domain–specific knowledge base; interrelationships between notes are computed from the relations between those concepts.

TINT represents specifications of note types and their attributes in LOOM, which has a formal semantics, classifies new concepts according to those semantics, and manages inheritance accordingly. In TINT, notes are related to each other via attributes and objects in an application's knowledge base; connections between notes are determined from connections between entities in the knowledge base. This enables the system to provide some very powerful tools for maintaining the organization of a system of notes, most notably classification–based reasoning for ensuring consistency and completeness in establishing connections, and BACKBORD–based browsing and retrieval aids for finding and inspecting them.

Capturing Meta–Knowledge. The BACKBORD specification by reformulation tool serves as a user interface to TINT, which SHELTER uses for capturing annotations of knowledge base entities. This in itself is a useful technological contribution, since retrieval of notes has long been an acknowledged weak point of hypermedia systems. By taking advantage of the formal representation of note types in the knowledge representation system, and of the formalized information that might be represented in attributes of notes, BACKBORD's powerful find/view/edit mechanisms can be used to locate notes.

Figure 6 illustrates the kinds of capabilities this approach is intended to support. It shows a situation in which a knowledge base developer has traced through one note to retrieve a set of others. In the top center window in the figure, several relations modelled in the knowledge base are highlighted in the text edit buffer to indicate that they have notes associated with them. The user has selected one of them, *item–sssr–ec*, and displayed an associated note (**Note 10022**) in the window on the left. This note instance contains pointers to fully–represented objects, like *Item–sssr–ec* and the *Expected Date* of **9/15/90**, as well as pointers to unstructured objects like the text provided for the *Problem* role.

Notes in TINT have types and inherit properties from higher types. One of the unique features of TINT, in fact, is a built–in domain–independent default type hierarchy; TINT has various special processing capabilities which application–specific note types can tap through inheritance. For example, **Note 10022** is an instance of an **Expectation Note**, a domain–independent note type used for reminders about states that are supposed to be satisfied at a later point in time. This note also qualifies as an application–specific specialization of that type, **Design Schedule Note**, and within that class has been further specialized to be a **Temporary Work–Around Note**. This denotes that it is a kind of note used to capture temporary changes to the model which are expected to be revised at a later point in time.

Working from **Note 10022**, the user has brought up a find/view/edit window (the bottom middle window), moved over various features from **Note 10022** (one of its types, and the values of its *For* and *Referent* relations), and generalized another feature (the *Expected Date* relation). The resulting description has then been used for retrieval, to get the list of related notes shown on the right in the figure. In SHELTER, lists like this one (which happens to show notes about relations modeled in the knowledge base for which there is some

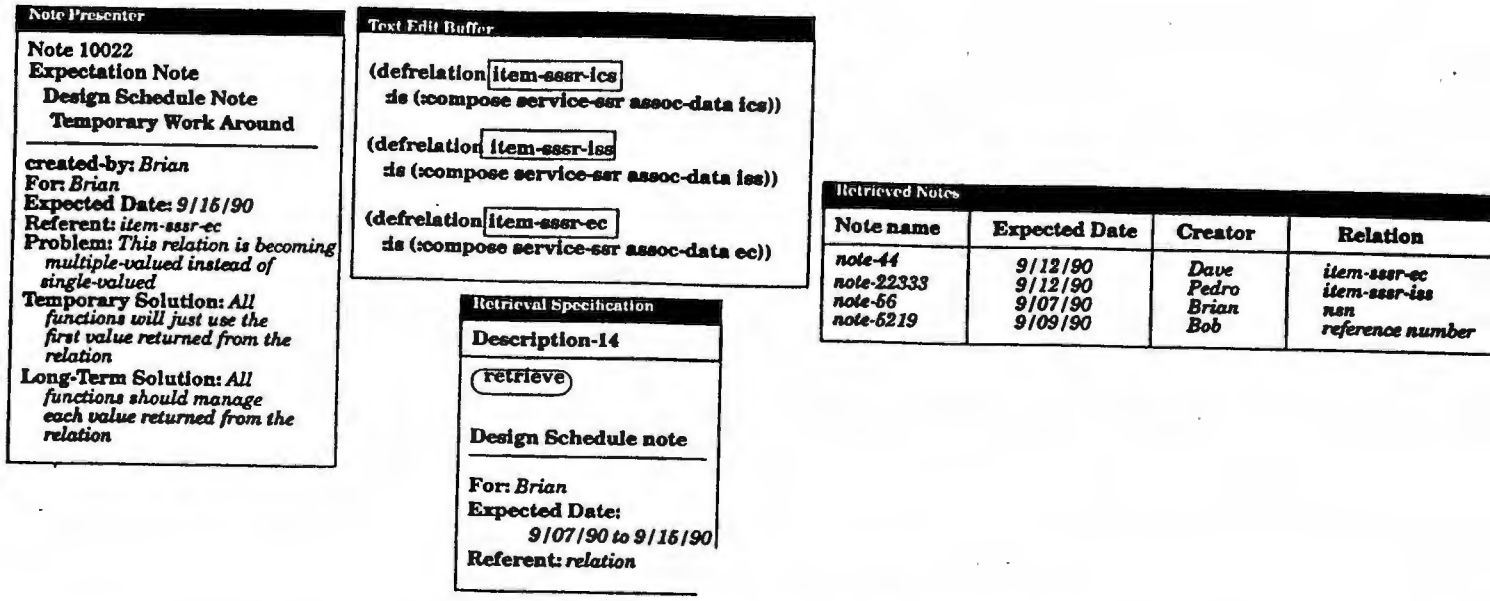


Figure 6 An Example Use of Find/View/Edit
to Browse Notes and Dynamically Construct Agendas

development change expected in a certain time frame) constitute dynamically constructed *agendas* of work to be performed. Since SHELTER will provide a shared environment for multiple developers, notes and agendas perform an important communication role.

TINT provides a ready mechanism whereby developers can annotate the knowledge base, both for their own benefit and as a mechanism of communicating with other developers. They may wish to do so to record the rationale behind representational decisions. They may want to explain the informal concept to which a formal model relates and/or highlight significant distinctions (for example, in logistical knowledge bases, it is not readily apparent from the formal specification that a "PCC–PLISN Pair" informally corresponds to the concept of a part being used for a particular purpose, as opposed to the part in isolation, e.g., a three–inch bolt on a landing gear as opposed to just a three–inch bolt.) They may want to guide or warn developers of constraints.

WORKING GROUPS IN THE KNOWLEDGE SHARING INITIATIVE

The desire to collaborate via knowledge sharing and reuse has arisen within a segment of the broad knowledge representation community that is interested in scaling up to larger systems, and views sharing and reuse of knowledge bases as a means to that end. Closely related to this is a concern for building embedded systems, in which knowledge representation systems support certain functionality rather than being ends in themselves.

In particular, our goal is to support researchers in areas requiring systems bigger than a single person can build. These include engineering and design domains, logistics and planning domains, and various "integrated modality" areas (e.g., multimedia interfaces). Researchers working on such topics need large knowledge bases that model complex objects; because those objects drive complex systems, they cannot be skeletons. Putting together much larger systems, of which various stand–alone systems being built today are just components, is an interesting challenge.

Creation of such knowledge resources requires community–wide effort. This engenders a need for agreed–upon conventions to enable us to build pieces that fit together. Eventually,

in pursuing the goal of large, shared knowledge bases as part of a nationwide information infrastructure, those conventions might become objects of study for the definition of more formal standards. (For those interested in the role of standards in computing infrastructure, Cargill 1989 is a useful entry point into the topic.) At present, however, the conventions are intended to support experiments in knowledge sharing among interested parties.

Starting with this section, the next part of this article will focus on operationalizing our vision by developing those conventions. Doing so is an essential precursor to larger aspects of our vision, such as libraries of ontologies, reasoning modules, and representation systems. This section will describe the activities of our four working groups on these foundation-laying activities. For each group, we will summarize the problem being addressed, the approach being taken, and the outcomes sought.

Interlingua

Interlingua is headed by Richard Fikes and Mike Genesereth, both of Stanford University.

Problem Formulation. The Interlingua working group is focused on the problems posed by the heterogeneity of knowledge representation languages. Specifically, in order to interchange knowledge among disparate programs (written by different programmers, at different times, in different languages, etc.), effective means need to be developed for translating knowledge bases from one specialized representation language to another. The goal of this group is to specify a language for communicating knowledge *between* computer programs (as opposed to a language for the internal representation of knowledge *within* computer programs). This language needs: (1) an agreed upon declarative semantics that is independent of any given interpreter; (2) sufficient expressive power to represent the declarative knowledge contained in typical application system knowledge bases; and, (3) a structure that enables semi-automated translation into and out of typical representation languages.

Approach. This group is specifying a language (*KIF*, for *Knowledge Interchange Format*) that is a form of predicate calculus extended to include facilities for defining terms, representing knowledge about knowledge, reifying functions and relations, specifying sets, and encoding commonly used nonmonotonic reasoning policies. The group is also conducting knowledge interchange experiments to substantially test the viability and adequacy of the language. The experiments focus on developing and testing a methodology for semi-automatic translation to and from typical representation languages and for using the interchange format as an inter-module communication language to facilitate interoperability.

Outcomes. The specification for Interlingua will evolve in a set of layers. The innermost layer will be a core, analogous to the primitives in LISP, providing basic representational and language extension functions. The next layer will provide idioms and extensions that make the language more usable, in analogy to the set of functions provided by Common LISP. This working group will be responsible for developing these specifications. Its output will be (1) a living document containing the current Knowledge Interchange Format specification, describing open issues, and presenting current proposals for modification; and, (2) a corpus of documented "micro-examples," using fragments of knowledge bases to illustrate how they translate into Interlingua and to point up open issues.

By the time this group has completed its work, we expect that the documented Interlingua specification will define a language in which sharing and reuse of the contents of individual knowledge bases can be accomplished by transmitting specifications using Interlingua as the medium for expressing those languages. The language will be oriented toward supporting translation performed with a human in the loop, and we expect that several prototype translation aids will be developed in the course of this work.

Knowledge Representation System Specifications

KRSS is headed by Bill Swartout of USC/ISI and Peter Patel-Schneider of ATT Bell Labs.

Problem Formulation. This group is concerned with specification, on a separate family-by-family basis, of the common elements within individual families of knowledge representation system paradigms. The intent is not to develop a "be-all, end-all" KR system. The group is not trying to develop one language that encompasses all approaches to KR; rather, it seeks to create specifications that incorporate the good features within families and, for each individual family, to develop a common version that is reasonably comprehensive yet pragmatic. An example of one such family is the set of KL-One descendents, i.e., object-centered languages with definitional constructs, such as CLASSIC, Loom, Back, SB-One, Omega, etc. To take an analogy, the effort should be viewed more as an attempt to develop a Common Lisp rather than a PL-1 or ADA. The analogy to Common Lisp is imperfect, however. Knowledge representation systems perform inference on the information represented within them; programming languages do not. Specifying what inferences should be performed is thus an additional issue in specifying a knowledge representation system.

Approach. The goal of this group is to allow system builders working within a family to provide a common set of features that have consensus within that paradigm, in addition to the augmentations that they regard as their research contribution. The resulting language will serve as a medium for sharing knowledge bases as well as a means for communicating ideas and issues among researchers. Potential benefits are the ability of users to more readily convert between systems and to borrow models originally built to run in other systems. The tradeoffs assumed by this group are the mirror image of those faced by Interlingua: they eliminate the problem of translating knowledge bases between systems, but require one to work within a given formalism in order to obtain that benefit.

Specifying a knowledge representation system poses the interesting challenge of specifying just what kinds of inference the system should perform. From Brachman and Levesque 1984, we know that if the system is expressive enough to be useful, the inferences that its reasoner draws will be incomplete. How should this incomplete inference be specified? The group's approach is to construct a layered specification. There will be an inner core of the language, which will consist of a few constructs. Because this inner core will have limited expressivity, it will be possible to perform complete inference on it. Another layer, the outer core, will be built on the inner core. This layer will significantly extend the expressivity of the language, but inference on it will be incomplete. To specify the inferences that should be performed, we will provide an abstract algorithm for drawing just those inferences that are required by the specification. Implementors of the specification must provide a reasoner at least as powerful as the one specified by this algorithm.

Outcomes. The group is seeking to develop a published specification and at least one initial implementation of a common language.

External Interfaces

The External Interfaces working group, co-chaired by Tim Finin of the Unisys Center for Advanced Information Technology and Gio Wiederhold of Stanford University, is focused on interfaces which provide interoperability between a knowledge representation system and other software systems.

Problem Formulation. The time is ending when an intelligent system can exist as a single, monolithic program that provides all of the functionality necessary to do a complex task. Intelligent systems will be used and deployed in environments that require them to interact

with a complex of other software components. These components will include conventional software modules, operating system functions, database servers, and other intelligent agents. There is a strong need to develop standard interface modules and protocols to make it easier to achieve this interoperability. The working group is concerned with three aspects of this problem: providing interoperability with other intelligent agents, with conventional (e.g., relational) database management systems, and with object-oriented database systems.

Approach. In order to provide run-time interoperability between knowledge representation systems, we need a language or protocol by which one system can pose queries or provide data to another. We have begun the specification of such a language, KQML (Knowledge Query and Manipulation Language). Our intent is that KQML will be to knowledge representation systems what SQL has become to database management systems — a high-level, portable protocol for which all systems will provide interfaces. Our current specification is organized as a protocol "stack" in which the lowest, information conveying layer is based on the Interlingua. Higher layers in this stack provide for modality (e.g., assert, retract, query), transmission (e.g., specifying the recipient agent or agents) and complex transactions (e.g., the efficient transmission of a block of data).

The integration of AI and DBMS technologies promises to play a significant role in shaping the future of computing. As noted by Brodie 1988, this integration is crucial not only for next-generation computing, but also for the continued development of DBMS technology and for the effective application of much of AI technology. This is due to the need for: (1) access to large amounts of existing shared data for knowledge processing; (2) efficient management of data as well as knowledge; and, (3) intelligent processing of data. The working group is studying the many existing interfaces between knowledge representation systems and relational databases (e.g., McKay, Finin, and O'Hare 1990), and is attempting to develop specifications for a common one. The primary issues here are the various ways in which the data in the databases can be best mapped into the knowledge representation objects.

The third task of the group is looking at interfaces between KR systems and object-oriented databases. The goal here is to use an OODB as a substrate under the knowledge representation system to provide a persistent store for knowledge base objects. This is exploratory work, but the potential long-term benefits are significant: (1) building and managing knowledge bases much larger than the current technology will support and (2) providing controls for transactions and concurrent access to knowledge bases at an object level.

Outcomes. The External Interfaces working group is concentrating on the development of the KQML protocol as its first goal. They hope that an early implementation will be used to help build testbeds for several distributed, cooperative demonstration systems. With regards to database interfaces, several working group members are attempting to integrate existing models for interfaces between knowledge representation systems and relational databases, and to produce a specification of a common one. The working group is also planning an experiment in which a simple interface will be built to allow an existing OODB to be used as a substrate under one of the representation systems being investigated by the KRSS working group. This will provide a feasibility test and generate data for further exploration.

Shared, Reusable Knowledge Bases (SRKB)

SRKB is headed by Tom Gruber of Stanford University and Marty Tenenbaum of EITech, Inc.

Problem Formulation. This group is working on mechanisms to enable the development of libraries of shareable knowledge modules and the reuse of their knowledge-level contents. Today's knowledge bases are structured as monolithic networks of highly interconnected symbols, designed for specific tasks in narrow domains. As a result, it is difficult to adapt existing knowledge bases to new uses or even to identify the shareable contents. To enable

the accumulation of shareable knowledge and the use of this knowledge by multiple systems, we need a means for designing composable modules of knowledge. The working group is chartered to identify the barriers to the building and use of shared knowledge modules, characterize potential approaches to overcoming these barriers, and conduct experiments exploring mechanisms for knowledge sharing.

Approach. The working group supports three kinds of activities. One is the identification of important research issues for knowledge sharing, including problems of methodology (e.g., multidisciplinary, collaborative knowledge base design) as well as engineering problems (e.g., scalability, shareability). A second activity is the development of ontologies that define terminology used to represent bodies of shareable knowledge. This includes both identifying bodies of knowledge worth the effort to represent formally and make shareable, and defining coherent sets of terms that characterize the ontological commitments and representational choices for modeling these bodies of knowledge. A third activity is the coordination of collaborative experiments in knowledge sharing, in which multiple research groups attempt to share and use each other's knowledge bases (e.g., libraries of device models). Some experiments will evaluate the use of ontologies as a mechanism for sharing (i.e., for modularity and composability of knowledge modules and for specification of their contents).

Outcomes. To these ends, the working group is concentrating on three objectives. The first is a survey of the state of the art in research on knowledge sharing and reuse, which identifies the techniques currently being explored and recommends research on the critical problems to be solved. A second outcome is a set of results from the collaborative experiments on knowledge sharing, including the ontologies used for each experiment and lessons learned about tools and methodologies for developing them. An immediate subgoal for this outcome is to develop a mechanism for representing these ontologies in portable form, building on the work of the other three working groups. The third, more long-term, objective is to develop a suite of exemplary shared ontologies and knowledge bases using them. These will serve as research testbeds, playing a role analogous to the E. Coli. bacterium in biology: a well understood, yet complete example upon which to perform a variety of experiments.

COMPARISON TO OTHER VISIONS

The model of knowledge-based systems which we have just described bears significant relationships to other notions which have been offered.

One recent example is MCC's Cyc Project (Guha and Lenat 1990), which seeks to develop an extremely large knowledge base of commonsense knowledge under which all applications would be built. The Cyc Project provides a language, Cyc-L, for implementing their ontology and for application-specific knowledge base development. Because its scope is so broad, Cyc represents one extreme in the range of efforts compatible with the model we propose. In our scheme, Cyc's knowledge base could be one very large entry in the library of components (or perhaps it might be broken down into several smaller modules or "microtheories"). Its implementation language, Cyc-L, would be one of the entries in the library of representation systems. If one chose to build a system entirely within Cyc, our model of the development process and theirs are largely consistent. If one wishes to go outside of Cyc, our model is complementary. Potential users might use our knowledge interchange format to translate other knowledge bases into Cyc-L or, conversely, to translate Cyc's knowledge into some other representation system. Alternatively, they might use external protocols to access a module built in Cyc, while building other modules through other means.

If successful, our approach would help make the knowledge in Cyc accessible even to those system developers who, for one reason or another, do not choose to use the whole knowledge base of Cyc or represent their systems in Cyc-L.

On the other hand, the model we propose leaves room to differ significantly from the philosophy behind Cyc. Among other things, it allows development of large systems without having to commit first to a particular knowledge representation formalism; users do not even have to commit to homogeneity. Furthermore, this approach also allows for the development and use of ontologies and knowledge bases under a more conservative, bottom–up development model, as an alternative to committing to a large, broad knowledge base. Thus, an alternative use of this model aims for the evolution of smaller, topic–specific ontologies intended for sharing by specialized communities. (Over time, it is possible that these topic–specific ontologies would grow and merge with others, so that an eventual end result might be a broadly encompassing ontology like that sought by the Cyc effort.)

Our vision owes a great deal to Mark Stefik's view of knowledge bases as the next mechanism of information exchange (Stefik 1986). Stefik was interested, as are we, in both how systems of the future will be built and what uses will be made of them. He suggested that expert systems were parts and precursors of a new *knowledge medium*, a set of interactive multidisciplinary tools for information manipulation: "A knowledge medium based on AI technology is part of a continuum. Books and other passive media can simply store knowledge. At the other end of the spectrum are expert systems which can store and also apply knowledge. In between are a number of hybrid systems in which the knowledge processing is done mostly by people. There are many opportunities for establishing human–machine partnerships and for automating tasks incrementally." In these interactive tools, the computer provides storage, retrieval, and group communications services, as well as languages and tools that enable precision and explicitness in manipulating knowledge about a topic.

In producing the new knowledge medium, Stefik argued that expert systems should be viewed as complex artifacts, like automobiles or airplanes, which are assembled from highly sophisticated materials and subassemblies. A marketplace that supports the specialized suppliers of component technologies benefits the manufacturer of the end product. The marketplace provides economies of scale which make it possible to develop component technologies of a quality and sophistication that would be unaffordable if the costs had to be borne entirely within a single, built–from–scratch, one–of–a–kind product development effort. Thus, by analogy, Stefik concluded that, "The 'goods' of a knowledge market are elements of knowledge.... To reduce the cost of building expert systems, we need to be able to build them using knowledge acquired from a marketplace. This requires setting some processes in place and making some technical advances."

In particular, Stefik urged combining work on expert system shells with work on creating standard vocabularies and ways of defining things in terms of primitives. This suggestion is very similar to the notion of ontologies proposed in this article. However, Stefik questioned (as do we) the feasibility of relying entirely on the development of standard vocabularies or ontologies for domains. The key to their effectiveness lies in how those ontologies are analyzed, combined, and integrated to create large applications. Tools and methods are needed to support that process. Our vision seeks to extend Stefik's by trying to further define the process, as well as the supporting tools and methods.

Our particular operationalization also has some kinship to the architecture of the national information infrastructure envisioned by Kahn and Cerf for the Digital Library System (Kahn and Cerf 1988). Their vision of the library of the future consists of information collections with some analogy to Stefik's knowledge media. The Digital Library System provides a distributed network architecture for accessing information. The architecture contains database servers, various accounting and billing servers, servers to support placing knowledge into the library and extracting knowledge from it, and servers for translating knowledge flowing to/from the library into different forms. The Digital Library System suggests a model for how our vision of development, distribution, and dissemination of knowledge based systems might be realized

in the future. At the same time, our vision proposes supporting technology — e.g., representation systems and translators — that could help realize the Digital Library System.

CONCLUSION

Attempting to move beyond the capabilities of current knowledge based systems mandates knowledge bases that are substantially larger than those we have today. However, representing and acquiring knowledge is a difficult and time–consuming task. Knowledge acquisition tools and current development methodologies will not make this problem go away, because the root of the problem is that knowledge is inherently complex and the task of capturing it is correspondingly complex. Thus, we cannot afford to waste whatever knowledge we do succeed in acquiring. We will be hard pressed to make knowledge bases much bigger than we have today if we continue to start over from scratch each time we construct a new system. Building qualitatively bigger knowledge–based systems will be possibly only when we are able to share our knowledge and build on each other's labor and experience.

The time is ripe to start building the infrastructure for integrating AI software at the knowledge level, independent of particular implementations. Today there is a significant body of ongoing work in AI and application domains on *pieces* of the problem, such as basic knowledge representation, knowledge acquisition tools, task–specific architectures, and domain–oriented, multi–use models. What is lacking is an integrating framework: a means for describing, connecting, and reusing knowledge–based technology.

Addressing these concerns will enable development of much larger scale systems, structured in a fashion that facilitates their development, maintenance and extension. It will eliminate barriers to embedding of AI components in larger, mostly conventional software systems. This will lead to a great expansion in the range of applicability for AI technology, which in turn will greatly enhance its utility and significantly expand the commercial marketplace.

This article has attempted to articulate a vision of the necessary knowledge representation systems technology, and of a path to achieving it, and has argued that progress in this area will dramatically change for the better the way that knowledge–based systems are built and the way they are perceived by their users. Central to that vision is the notion of establishing an information infrastructure for promoting sharing and reuse of knowledge bases in development and application of large, enterprise–level software systems.

ACKNOWLEDGMENTS

This effort is supported by NSF Grant IRI–900–6923, NASA–Ames Contract NCC 2–520, and a cooperative agreement between USC / ISI and the Corporation for National Research Initiatives. Portions of this paper were extracted from *Enabling Technology for Knowledge Sharing*, to appear in the Fall, 1991 edition of **AI Magazine**; co–authors on that paper were Richard Fikes, Tim Finin, Tom Gruber, Ramesh Patil, Ted Senator, and William R. Swartout. Brian Harp and Pedro Szekely have played important roles in the SHELTER effort described in other sections of this paper.

BIBLIOGRAPHY

Bennett, J.S. 1984. ROGET: Acquiring the Conceptual Structure of a Diagnostic Expert System. In Proceedings of the IEEE Workshop on Principles of Knowledge–Based Systems, 83–88. Denver, Colorado: IEEE.

Brachman, R.J., and Levesque, H.J. The tractability of subsumption in frame–based description languages. In Proceedings of AAAI–84, 34–37. Austin, Texas: AAAI.

Brodie, M. 1988. Future Intelligent Information Systems: AI and Database Technologies Working Together. In *Readings in Artificial Intelligence and Databases*, eds. J. Mylopoulos and M. Brodie, 623–641. San Mateo: Morgan Kaufmann.

Cargill, C.F. 1989. *Information Technology Standardization: Theory, Process, and Organizations*. Maynard, MA: Digital Equipment Corporation Press.

Chandrasekaran, B. 1983. Toward a Taxonomy of Problem Solving Types. *AI Magazine* 4(4): 9–17.

Chandrasekaran, B. 1986. Generic Tasks in Knowledge-Based Reasoning: High Level Building Blocks for Expert System Design. *IEEE Expert* 1(3): 23–30.

Gruber, T. R. 1991. An Experiment in the Collaborative Development of Shared Ontology, Technical Report, Knowledge Systems Laboratory, Stanford University.

Guha, R.V.; and Lenat, D.B. 1990. Cyc: A Mid-Term Report. *The AI Magazine* 11(3): 32–59.

Harp, B. and Neches, R. NOTECARDS: An Everyday Tool for Aiding in Complex Tasks. In J. Sullivan & S. Tyler (Eds.), *Proceedings of the First Asilomar Workshop on Intelligent Interfaces: Elements and Principles of Intelligent Interfaces*, 1988. Also available as USC/ISI Technical Report ISI/RS–88–204.

Kahn, G., Nowlan, S., and McDermott, J. 1984. A Foundation for Knowledge Acquisition. In Proceedings of the IEEE Workshop on Principles of Knowledge–Based Systems, 89–96. Denver, Colorado: IEEE.

Kahn, R.E., and Cerf, V.E. 1988. An Open Architecture for a Digital Library System and a Plan for Its Development, Volume I: The World of Knowbots, Technical Report, Corporation for National Research Initiatives.

Lai, Kum–Yew, and Malone, Thomas, Object Lens: A 'Spreadsheet' for Cooperative Work, in the *Proceedings of Computer Supported Cooperative Work Conference*, September, 1988.

McDermott, J. 1990. Developing Software is like Talking to Eskimos about Snow. In Proceedings of AAAI–90, The Eighth National Conference on Artificial Intelligence, 1130–1133. Boston, MA: AAAI.

McKay, D., Finin, T., and O'Hare, A. 1990. The Intelligent Database Interface. In Proceedings of the AAAI–90, the Eighth National Conference on Artificial Intelligence, 677–684. Boston, MA: AAAI.

Pan, J.Y.–C.; Tenenbaum, J.M.; and Glicksman, J. 1989. A Framework for Knowledge–Based Computer–Integrated Manufacturing. *IEEE Transactions on Semiconductor Manufacturing* 2(2): 33–46.

Steels, L. 1990. Components of Expertise. *The AI Magazine* 11(2): 29–49.

Stefik, M. 1986. The Next Knowledge Medium. *The AI Magazine* 7(1): 34–46.

Waterman, D.A. 1986. *A Guide to Expert Systems*. Menlo Park, CA: Addison–Wesley.

Yen, J., Neches, R., DeBellis, M., Szekely, P., and Aberg, P. BACKBORD: An Implementation of Specification by Reformulation. In J.W. Sullivan and S.W. Tyler (Eds.), *Intelligent User Interfaces*. New York: ACM Press, 1991, pp. 421–444.

Knowledge Adaptation: a Means of Knowledge Acquisition*

Alvaro de la Ossa
Research Group on Artificial Intelligence (Expert Systems)
Dept. of Computer Science
University of Kaiserslautern
PO Box 3049
D-6750 Kaiserslautern
Federal Republic of Germany
e-mail: delaossa@informatik.uni-kl.de

ABSTRACT

Knowledge Adaptation is an alternative means of knowledge acquisition from previous experience in similar application domains. Adapting knowledge from an old expert system in some domain means transforming rules into rules useful for problem solving in a new, similar domain. The transformation follows from an analogical mapping of the situation described in a rule to situations in the new domain. This paper describes the Knowledge Adaptation approach to knowledge acquisition. We use diagnosis as the exemplar domain task, MOLTKE[1] as the computational testbed, and CNC-machines[2] as the exemplar application domain.

1. INTRODUCTION

Our research group has developed MOLTKE, a workbench for the construction of expert systems for technical domains. The main domain tasks have been diagnosis and configuration, and components for our workbench on other tasks as process planning, configuration, hypermedia, etc., are being developed. In the domain of diagnosis, exemplar application domains have been CNC-machines, 3D-CNC measurement machines, and heterogenoues computer networks. In this work we use the first application domain for illustration.

The CNC-machine producer configures *different machine versions* using parts from a

* This work is being developed within project X9 (Learning and Analogy in Technical Expert Systems) of the special research area SFB-314 (Artificial Intelligence — Knowledge-Based Systems). The author is a graduate student holding a fellowship from the German Service for Academical Exchange (DAAD, Deutscher Akademischer Austauschdienst) and support from the University of Costa Rica.

1 For **MO**dels, **L**earning, and **T**emporal **K**nowledge in **E**xpert systems for technical domains

2 For **C**omputerized **N**umerical **C**ontrol; see, e.g., (Althoff, Maurer, and Rehbold, 1990)

common library. The knowledge adaptation problem consists in transforming knowledge from an old machine's expert system to be used within the same domain task on a newer, *similar* machine. Adapting knowledge involves reasoning about differences and similarities between both machines and using knowledge adaptation strategies accordingly.

The knowledge adaptation scenario arises from the need of *reducing the effort* invested *in the knowledge acquisition task* during the early stages of expert system development. Knowledge adaptation can be seen as a means of *making knowledge reusable for solving similar situations within the same domain task.*

The paper is organized as follows. Chapter 2 discusses the knowledge adaptation problem. Chapter 3 gives a brief description of a MOLTKE diagnostic expert system and of the CNC-machines domain. Chapter 4 deeply discusses each step of the knowledge adaptation process. For each step, the expected role of the human experts during knowledge adaptation, as well as the strategic knowledge acquisition techniques used, are described. Chapter 5 presents the acquisition and refinement of strategies for improving the knowledge adaptation process. Finally, chapter 6 discusses the relations of this with other work, and describes the state of our research and our next future plans.

2. THE KNOWLEDGE ADAPTATION PROBLEM

A typical flaw of expert systems, due in part to the intimate dependency on the application domain, is the difficulty to make knowledge *reusable* in other similar domains. An expert system might solve problems in a particular application domain, but when this is modified, even lightly, the system may fail to solve problems with acceptable reliability[3].

Sometimes, totally new knowledge bases must be developed for a new domain. Others, the human expert and the knowledge engineer work hard and long together in the task of extracting from the old expert system knowledge that can be useful for the new domain, adapting it accordingly, and incorporating it into a new knowledge base.

But no solution towards the automatization of this process can be found explicit in the literature. There are attempts to define means of making the domain modeling task more flexible (Morik, 1987, 1988; Schreiber, Wielinga, and Breuker, 1991). In our oppinion, the enormous costs due to knowledge acquisition, normal during early stages of system development, cannot be drastically reduced by means only of flexibility in modeling. A learning system must be able of *explicitly adapting its knowledge to a changing domain.* Without this ability, learning is limited to a static view of the domain.

In technical domains in particular, the application domain is mostly described by a model of the physical world, adequate for a particular domain task, e.g., diagnosis (Struss, 1988). The model allows for causal reasoning about structure and behavior. But models alone lack of means of associating experience in the domain task with causality

3 *Reliability* is used here to mean empirical acceptability criteria set by the human expert

relationships in the model itself. Moreover, models are normally incomplete and too simplifying. Modifying the physical world requires of modifying the model accordingly. The lack of association from experience to causality leads, after a modification, to the loss of experience gained on problem solving for the previous state of the physical world.

On the other side, the empirical acquisition of knowledge in the domain is mostly justified by the human expert's advice, due to the inability of most empirical approaches to provide by themselves, using background knowledge, for well-founded explanations of the acquired knowledge.

Thus, a bridge from the model of the physical world to the experience implicit in empirical knowledge and viceversa must be made available, that can allow for justifying the automatic adaptation of knowledge from the old expert system to cope with the domain task on the new physical world.

2.1. Overview of the Knowledge Adaptation Approach

Given an already developed diagnostic expert system for some machine and a new, functionally *similar* but structurally different machine, we wish to adapt the old system's knowledge to be (re)used for diagnosis on the new machine.

Our approach to knowledge adaptation identifies knowledge suitable for adaptation by using a *similarity-based* approach. Two machine parts are considered similar if they belong to a same parts class and their *functionality* descriptions are similar. A part's functionality and the similarity metric used are described later in section 4.3.1.

A simple *model-based* reasoning mechanism is used to *analogically* map a diagnostic situation from the old machine into a new, *plausible* situation for the new machine. The analogical mapping is approached on two levels. First, following domain-independent mechanisms, a situation is explained on the domain model. Then, domain-dependent features are used to complete the analogical mapping. The plausibility of the new situation is verified through simulation automatically guided by background knowledge[4].

The knowledge adaptation component communicates with the performance element of the expert system to generate a hypothesis for the new, plausible situation. If such a hypothesis is found, the resulting *rule* (i.e., diagnosticSituation → hypothesis) must be incorporated into the new expert system. The rule can be incorporated if it results *consistent* with the current state of the new expert system.

The whole process, from the identification of similar parts in the old machine to the incorporation of an adapted rule in the new expert system is called an *adaptation case*. Adaptation cases are generalized to *adaptation rules*, that represent strategic knowledge for adaptating groups of diagnostic rules sharing a common general diagnostic situation and the same particular hypothesis.

---

4 See (Dutta, 1988) for a deeper discussion on justifying analogical transformation through the intensive use of background knowledge. Other approaches to verification use *experimentation*, as in (Carbonell and Gil, 1990; and Shen and Simon, 1989)

3. THE MOLTKE DIAGNOSTIC EXPERT SYSTEM

The workbench MOLTKE plays the role of computational testbed for our research. In this chapter, a brief description of the basic terminology and of the diagnostic task in MOLTKE is given, including some details relevant for our knowledge adaptation component. We illustrate with our exemplar application domain. Further details about the conceptual approach to diagnosis and methodological development of our workbench are found in the references mentioned through the description.

3.1. The Development of a MOLTKE Diagnostic Expert System

The diagnostic task is seen in MOLTKE as the combination of *classification* and *test selection*. Classifying means determining a rough or intermediate diagnosis for a given fault. Test selection is used to refine the current diagnosis until a final diagnosis is found. Diagnoses are organized in a *context graph*, where a node, called context, represents a diagnosis, and the links, refinements between contexts.

For knowledge adaptation purposes, the generation of the context graph is incremental, i.e., new contexts are generated only when they are needed for diagnosis or for knowledge adaptation. This has the advantage of reducing the time required for knowledge base generation for a new machine; it also allows the easy changing of the knowledge base in response to light modifications in the structure of the new machine[5].

In our expert system, diagnostic knowledge is structured *vertically*—in a heterarchy of *contexts*, and *horizontally*—separated in *ordering* and *shortcut* knowledge. Contexts represent knowledge about *failures* (*classification*), and ordering and shortcut knowledge, knowledge about the *diagnostic process* (*test selection*).

A *context* has a precondition (conjuntion or disjunction of conjunctions of symptom values) which, when satisfied under the current situation, causes the associated *diagnosis* to become *proven*. To every context a set of ordering rules and a set of shortcut rules are associated, as well as a *context interpreter*. The locality of ordering rules and the presence of a (local) interpreter allow for specialized strategy for test selection.

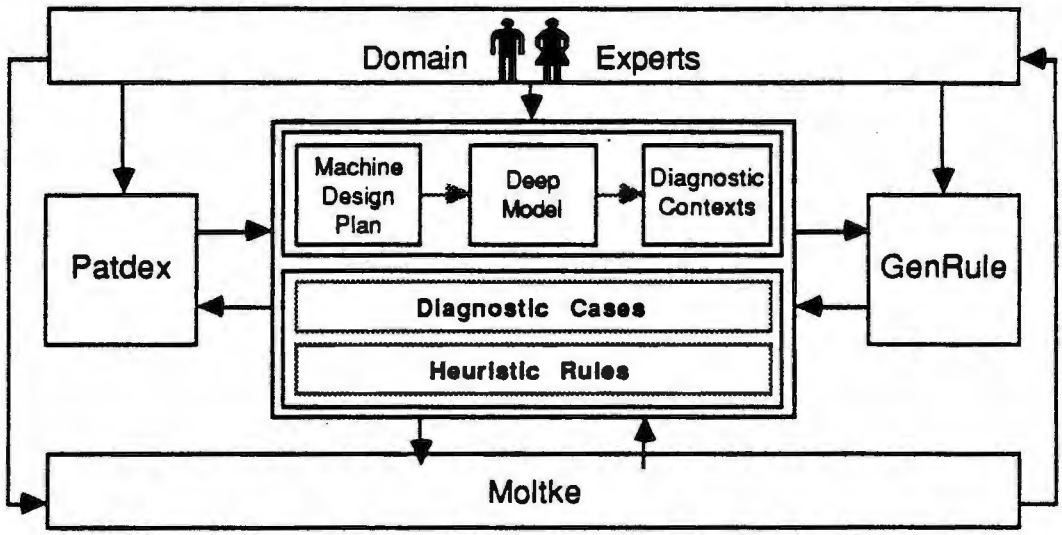
Definitions

Symptom Class — relates a name with a list of possible values (e.g., `valve` and `(open, closed)`)
Symptom instance — (or symptomvalue, or symptom) describes the state of a machine part (e.g., `(Valve27Y1 closed)`)
Situation — set of all symptoms; (partially) describes the actual state of the machine
Formula — stores the current binding of a symptom (a variable in the predicate calculus with a three-valued logic—`true`, `false`, `unknown`—used to evaluate the formulas in a *formula language*)
Test — determines the current value of one or more symptom instances
Ordering Rules — determine which test to execute next; the order of these rules determines the (potential) execution order of tests

5 For instance, after replacement of defectuous parts

Shortcut Rules — represent a relation between symptom values, and shorten the diagnostic process by deriving one or more (still unknown) symptom values from one or more (known) symptom values, thus eliminating the need of gathering their values through tests

Figure 1 below shows the overall process of expert system construction in MOLTKE.



— Figure 1 — The Construction of the Expert System —

3.1.1. Expert System Generation : Modeling and Simulation

Diagnostic knowledge stored in contexts is extracted automatically and directly from the machine's design plans. The automatic model and diagnostic knowledge compiler MAKE[6] (Rehbold, 1991) builds up a static, deep model of the machine's structure and behavior. The model is checked for consistency and the context heterarchy is built through a simulation process. For each context, its precondition and shortcut rules are generated. MAKE's output is a basic expert system, which may be later refined and extended[7].

A new version of MAKE, called ¡MAKE, is currently being implemented in the context of this research, which allows the incremental extension and modification of the context graph. ¡MAKE reduces the effort required for context graph generation by creating a library of subgraphs associated to machine parts according to their functionality.

In the machine's model, specific parts are described by classes of parts, which store mainly its *name*, *ports*, *behavioral* description (input-output rules relating ports), and *typical faulty behavior*[8]. *Primitive* parts cannot be decomposed. *Complex* parts store additionally their *subparts* and *connectivity* (physical connections between subparts). The structure is represented hierarchically. The behavior of complex parts is elaborated

6 For **M**odel-based **A**utomatic **K**nowledge **E**xtractor
7 By, e.g., a *service technician* or a machine's *engineer*, in the case of CNC-machines
8 For simplicity, only the model slots relevant for this discussion have been described

on the basis of their component parts.

iMAKE extends a part instance's description with its *functionality*. This is defined as the subset of behavioral rules that describe the actual part's behavior in the particular configuration in which it is placed. That is, a part's behavior describes all possible outputs from the part for all possible inputs (what is the part designed to do?), while functionality describes the outputs that are expected from the particular connections of the part in a particular machine (what does the part do in this machine?).

Once the model is built, iMAKE simulates it in search for diagnostic information that can be made explicit in MOLTKE's representation formalism. Two tasks are included in this process: definition of symptom and tests, and generation of contexts[9] and their rules. A symptom class is generated for each port of every primitive part instance in the machine. A test is generated for each symptom class. Later in MOLTKE, a symptom will be represented by a symptom instance and the actual value measured on the respective port.

During simulation, for each port on which a deviation from an expected output value can be found, a context is generated, and its name, precondition, and correction are defined. Intermediate diagnostic contexts (those that are not leaves in the graph) additionally contain shortcut rules, refinement links (to other contexts), and ordering rules.

The context heterarchy reflects the diagnostic expert's proceeding during diagnosis[10]. The search space is shortened by considering only those behavioral rules *relevant* for describing the *intended behavior* of the machine part[11]. A context class is generated for every *relevant* faulty output value (from the faulty behavior description of the part).

In our new conception of this process, iMAKE does not need to generate all the contexts for a given machine part if that task was solved before for other part (even from other machine) of the same class and with the same functionality conditions. iMAKE stores a newly generated part's context subgraph and associates it to the part's class and its particular functionality. Context subgraphs are later used as patterns for other parts of the same class and with the same functionality conditions, which only need to be consistently inserted in the whole context heterarchy.

3.1.2. Refinement and Extension of the Expert System's Knowledge

The basic MOLTKE-expert system can be refined, e.g., by modification of shortcut rules, inclusion of new ordering rules, etc. This task is carried out in joint work by the expert and the knowledge engineer. The expert elaborates flow diagrams of test execution order, which are traduced by the knowledge engineer into ordering rules and input into the corresponding contexts. The expert can edit the machine model as well, as the context graph, using graphical tools developed specially for that purpose (based on Smalltalk's Browser interface[12]).

---

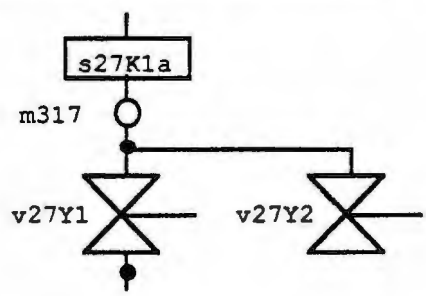
9 Additionally, *corrections* (to the faults) must be also generated, but this is not relevant here
10 We found this to be true for all technical domains modeled so far
11 This is MAKE's terminology, which agrees with our definition of *functionality*
12 Smalltalk is the implementation language of MOLTKE

Moreover, the system acquires, organizes, and refines diagnostic experience through its learning component. Experiential knowledge is case-based modeled. An *ordered set of symptoms* associated to a *hypothesis* defines a *diagnostic case*. Two subsystems cooperate to enhance the diagnostic task. PATDEX/2[13] (Althoff, De la Ossa, Maurer, Stadler, and Weß, 1989; Althoff and Weß, 1991a) is a case-based reasoning system that retrieves the most similar case to the current situation. Similarity is defined in terms of the symptoms in the situation. The hypothesis to the retrieved case is mapped to the current situation. PATDEX/2 acts interactively on-line to enhance the diagnostic process. The other subsystem, GENRULE[14] (Althoff and Traphöner, 1990; Althoff, 1991), acts off-line by compiling heuristic generalization rules from diagnostic cases. Those rules, which describe partial shortcut rules, are given a statistically-determined certainty factor.

3.2. A Sample Application Domain : CNC Machine Centers

A CNC machine center is a production tool machine that gives some raw material a desired form and precise measure through controlled relative movement; it has a magazine of tools from where it can choose and seize one to work with on a piece of material. Some of its operations on the material are drilling, milling, rubbing, tap sharpening and cutting, sawing, etc. The basic components of a machine center are rack, sliding carriage, guides, and drive or propulsion. Attached are other facilities for control, regulation, changing, etc., of tools and material. A large knowledge base for fault diagnosis of a machine center has been developed by our research group[15].

Examples of different configurations leading to functionally similar machines can be drawn ranging from very high abstraction levels in the machine's structure, e.g., using a tool claw arm with finger elements instead of a claw arm with plier claw and rotation, to very low levels, e.g., using one of two different valves in a hydraulic or electrical element. Figure 2 shows a sample machine part design plan. We show its model of structure and behavior and a sample diagnostic context from our knowledge base.



— Figure 2 — Sample Design Plan of a Complex Machine Part —

13 For **PAT**tern-Directed **EX**pert system
14 For **GEN**erator of empirical MOLTKE **RULE**s
15 Some parts (e.g., the tool changer, etc.) have been modeled, not the whole CNC machine, but the modeled domain is complex enough for our purposes; knowledge bases from other technical domains have also been developed, and the results of experiments with them have given fruitful feedback during the development of our workbench

Instance **v2s27Y1-2** of class **Valve2Selection** is entered to the system using a graphic model editor. After modeling, the resulting description for the instance and its subparts looks like follows:

```
ComplexPart Class #Valve2Selection
      ports:          #((lever mechanical in low) (currentIn ac in medium)
                        (currentOut ac in medium) (slide1 mechanical out low)
                        (slide2 mechanical out low))
      subparts:       #((Switch+ s low) (MeasuringPoint m low)
                        (Valve v1 low) (Valve v2 low))
      connections:    #((self currentIn currentIn s) (s currentOut currentIn m)
                        (m currentOut currentIn v1) (m currentOut currentIn v2)
                        (v1 currentOut currentIn self) (v1 slide slide1 self)
                        (v2 currentOut currentIn self) (v2 slide slide2 self)).
Valve2Selection Instance #v2s27Y1-2
      location:       'toolChanger'
      subparts:       #((s s27K1a 214) (m m317 214) (v1 v27Y1 275) (v2 v27Y2 276)).
PrimitivePart Class #Switch+
      ports:          #((current1 dc inOut medium) (current2 dc inOut medium)
                        (lever mechanical in low))
      behavior:       #((lever = unshifted) (current1 = X) -> (current2 = 0))
                        (lever = shifted) (current1 = X) -> (current2 = X))
                        (lever = unshifted) (current2 = X) -> (current1 = 0))
                        (lever = shifted) (current2 = X) -> (current1 = X)))
      failures:       #((noContact (((current1 = X) -> (current2 = 0))
                                    ((current2 = X) -> (current1 = 0))))).
```

During simulation, the context graph is generated. Following is a sample context for **Spindle**, a part containing **v2s27Y1-2** as one of its subparts:

```
#SpindleStopNotReached
precondition:
      (= SpindleStop SpindleStop SpindleStopNotReached)
shortcutRules:
      (if (and (IsPreconditionOf SpindleStopNotReached) (= Valve Valve27Y1 Closed) )
       then IOSstatusOUT32 logic1 factor "total")
      (if (and (IsPreconditionOf SpindleStopNotReached) (= IOStatus IOStatusOUT32 1) )
       then IOStatusIN1112 logic1 fator "total")
orderingRules:
      (if (true) then Valve27Y1 test)
      (if (= Valve27Y1 closed) then OilConductsOrientedSpindleStop test)
      (if (= Valve27Y1 open) then IOStatusOUT32 test)
      (if (= IOStatus IOStatusOUT32 0) then IOStatusIN1112 test)
      (if (= IOStatus IOStatusIN1112 0) then VoltagePotential361362 test)
```

3.2.1. CNC-Machines Diagnosis

A failure in the CNC-machine is reported by the CNC *Input/Output Card* , which sends a *machine failure code*, directly associated to machine parts normally on very high abstraction levels in the machine's structure. A diagnostic situation is initiated when a machine failure code is received from the CNC card. This code has the advantage of restricting the search for a diagnosis on a particular machine part and its subparts. Then, during diagnosis, the expert system enters a *establish-and-refine* cycle of testing for symptoms and refining to more particular diagnostic contexts.

4. KNOWLEDGE ADAPTATION

The knowledge adaptation task is seen as an incremental process in which diagnostic rules and cases from the old expert system are transformed into *rules* for the new system. A rule has in its left-hand side a diagnostic situation (an ordered set of symptom values) and in its right-hand side a hypothesis[16] .

The transformation of rules leads to rules whose situations are *plausible* in the new machine[17] and whose hypotheses shall be proved by the new expert system. The main adaptation strategy in our approach builds an analogy of the situation of a rule from the old machine onto the new machine. The hypothesis for the new situation is obtained by simulating the new situation in the new expert system.

The approach foresees the acquisition, extension, and refinement of adaptation strategies by exploiting the results of successful rule adaptation to improve later performance. Knowledge adaptation is seen as a process of proposing new diagnostic rules and learning from their verification. A case memory for adaptation cases is currently being implemented in the context of this research. Adaptation rules (strategies) are stored associated to particular machine part classes and their faults. A memory architecture inspired in Kolodner's is being used as a model (Kolodner 1983a, 1983b, 1989).

Besides the feedback from the human expert upon accepting or rejecting adapted rules, the expert can also play the role of a tutor by entering manually prepared adaptation cases, considered as positive examples of a particular adaptation strategy. In this case, the expert must provide the system with an indication of how the analogy that leads to the adaptation is originated, in the form of a causal explanation on the model of the machine. This advice can be used later as an alternative strategy to analogy recognition if this is not possible on the machine's model level. A detailed description of the generation of causal explations of a rule is given later.

In the next sections we analyze the kinds of knowledge in our expert system, compel the use of the terms *case* and *rule*, and describe the knowledge acquisition process.

4.1. Causal and Empirical Knowledge in our Expert System

Causal or *background* knowledge describes the application domain (the machine and its diagnosis), while *empirical* knowledge describes experience acquired during diagnosis or directly from the expert. Causal knowledge is acquired directly from the design plans of the machine. This knowledge is stored initially by ¡MAKE in a *static* knowledge base containing the machine's *model* (knowledge about the particular domain) and the *context graph* (diagnostic knowledge in the particular domain).

Empirical knowledge can be acquired either directly from the *domain expert*, through

16 The use of the terms *rule* and *case* is made clear later in section 4.2.

17 The term *plausibility* is here used to mean that the situation may physically occur in the machine, or in other words, that the symptoms in the situation are not inconsistent with the machine's model

actual *experience* during diagnosis (MOLTKE and PATDEX), or by automatic generation starting from diagnostic cases (GENRULE). Four kinds of empirical knowledge are identified:

Diagnostic cases : successful diagnoses, either actual, or known to the expert and given directly to the system. They are represented as follows:

$$S_1, S_2, \ldots, S_n \rightarrow \mathcal{D}h$$

where S_k ($k = 1, \ldots, n$) are known symptom values, and
$\mathcal{D}h$ is a diagnostic hypothesis (the name of a certain context).

Test selection cases : selection of tests, given a particular diagnostic situation, leading to gathering useful symptom values during actual diagnosis. They are represented as follows:

$$S_1, S_2, \ldots, S_n \rightarrow \mathcal{T}h$$

where S_k ($k = 1, \ldots, n$) are known symptom values, and
$\mathcal{T}h$ is a test hypothesis (the name of a certain test).

Ordering rules : empirical determination of the order in which tests must be executed in order to refine the diagnostic hypothesis for a given context. They are represented the same as test selection cases with the difference that test selection cases are stored in a case memory and ordering rules in the particular expert-given order within a diagnostic context.

Shortcut rules : empirical determination of one or more (still unknown) symptom values given a particular diagnostic situation. This rules can be either directly given by the expert, or generated off-line by GENRULE, and are represented as follows[18]:

$$S_1, S_2, \ldots, S_n \rightarrow S_{n+1}, S_{n+2}, \ldots, S_m$$

where S_k ($k = 1, \ldots, n$) are known symptom values, and
S_l ($l = n+1, \ldots, m$) are (still unknown) symptom values.

4.2. Cases and Rules : What Should Be Adapted?

It is important to note the difference between cases and rules[19]. Cases are examples of real diagnostic situations and their solution. They contain a problem description (the situation), a solution (the hypothesis), and a justification for the solution, which should contain an explanation of how the solution was found, in terms of the situation.

In MOLTKE, the justification for the solution of a diagnostic or test selection case is the temporal order (sequence) in which symptom values were gathered through test execution. A case's situation $S_1, S_2, \ldots, S_n$ is actually given in this order: symptom S_k was gathered before symptom S_{k+1}. A more grounded justification is still missing in MOLTKE. Thus, for knowledge adaptation, we extend a case's justification to a more complete explanation, by abstracting the symptom values in the situation, in their order, on the

18 Shortcut rules are also generated by MAKE. But these rules are justified on the machine's model (they are called *total*); they do not reflect empirical knowledge

19 A more extense discussion on cases and rules can be found in (Althoff and Weß, 1991b)

machine's model. This is done through a simulation process in which parts of the machine involved in the situation and their causal (structural and behavioral) relationships are identified.

Rules, on the other side, differ from cases in that they do not have a justification. Their interpretation is unique, while a case's interpretation depends on the purpose, the similarity with other cases, and so on. Ordering and shortcut rules in MOLTKE have no justification, or in other words, the justification is indeed empirical. Ordering rules are entered into the knowledge base (into particular contexts) by the knowledge engineer, traduced from flow diagrams of test execution order prepared by the human expert. The generation of a model-based justification of ordering rules, and furthermore, of their order, is, for the moment, still not considered, although this will imply a great reduction in knowledge acquisition effort.

We have so far tested the approach only with diagnostic and test selection cases. What makes ordering rules more complex is the fact that their justification must not only take into account each of a context's rules, but also identifying equivalent contexts in the new machine, as well as considering their sequence or order.

Finally, empirical shortcut rules' justification rests on the diagnostic cases from which they were abstracted by GENRULE. We still do not consider shortcut rule adaptation for one reason: after adapting diagnostic and test selection cases, GENRULE can be applied on the new expert system to generate new shortcut rules. It would be however interesting to approach their adaptation and compare the results with GENRULE's.

In the rest of the discussion we refer to diagnostic and test selection cases using the terms *rules* and *cases* without distinction. Following, we describe KALES[20], which is how we have named our knowledge adaptation approach. Along the discussion we describe the intervention of the experts. In chapter 5 we discuss the means of acquisition and refinement of knowledge to improve the adaptation strategy. Due to space limitations, representation and implementation details (particularly algorithms) are left out except where required for the discussion.

4.3. The Knowledge Adaptation Task

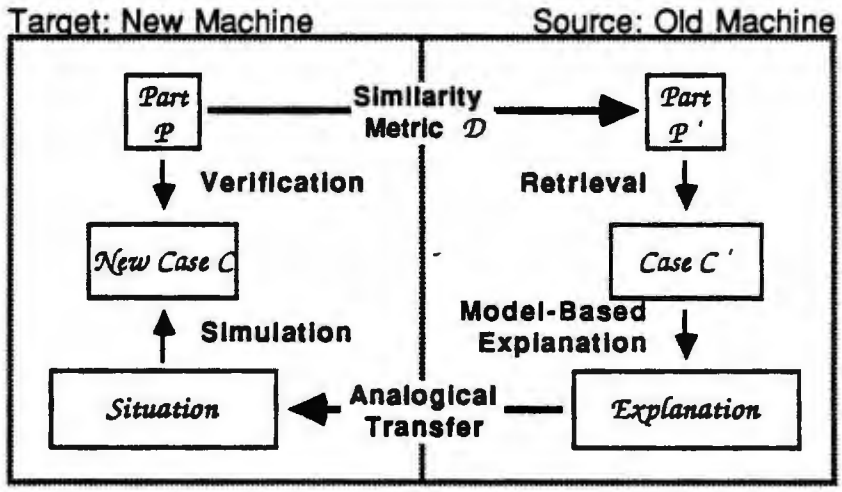
Knowledge adaptation is a source of knowledge acquisition for the new expert system. It is based in the main assumption that knowledge from the expert system for an old machine can be reused for diagnosis on a new machine if it is possible to find a *correspondnce* between both machines.

We assume that both machines are functionally similar, in the sense that they both serve a similar purpose, e.g., drilling, sawing, etc. But they are structurally different, that is, the internal structure, connections among subparts, and internal behavior of the new machine can be seen as a modification of the old machine. Finally, we assume that both machines' subparts are taken from a common library of components.

---

20 For Knowledge Adaptation in Learning Expert Systems

Knowledge adaptation is incremental. First, the process is normally carried out off-line, although on-line, interactive sessions of adapting rules for specific faults can be set with the experts. Second, a step in learning of knowledge adaptation strategies is done from one rule rather than from a set of them, although an incremental definition from sets of rules is possible. Figure 4 below gives a synopsis of the knowledge adaptation task.



— Figure 4 — The Knowledge Adaptation Task —

4.3.1. The First Step : Identification of Knowledge for Adaptation

Every time iMAKE generates the diagnostic contexts for parts of the new machine, KALES tries to adapt empirical knowledge for similar machine parts from the old machine. Thus, the first subtask of KALES is identifying those machine parts. This follows from a similarity-based approach. Recall that the functionality of a machine part $\mathcal{P}$ is defined as the subset of behavioral rules actually used in the particular location of $\mathcal{P}$ in the machine. Two machine parts $\mathcal{P}_1$ and $\mathcal{P}_2$ are *functionally similar* if they

(i) belong to the same parts class,
(ii) have the same number of input and output ports, and
(iii)

$$\mathcal{D}(\mathcal{P}_1, \mathcal{P}_2) := \frac{C_{12} - \mathcal{N}_{12}}{C_{12} + \mathcal{N}_{12}} > \epsilon \quad \text{, where}$$

C_{12} is the number of corresponding and $\mathcal{N}_{12}$ the number of noncorresponding rules in the functionality description of parts $\mathcal{P}_1$ and $\mathcal{P}_2$, and ϵ is some threshold value in [-1,1] (most likely close to 1), that can be interpreted as the minimum (empirically) acceptable similarity.

This is a still simple similarity metric. It can be extended, for instance, by weighting the relevance of rules in the functionality descriptions of both parts. We do not deepen the discussion on similary metrics here, we just propose a metric that can be later improved.

KALES searches for functionally similar parts of the old machine. The search is shortened by restriction (i) in our definition. This avoids the exponential nature of the search in the machine's structure hierarchy. With the list of retrieved old-machine parts, KALES accesses the case memory in search of cases in which the failure code or the diagnostic hypothesis refers to the parts in the list. The retrieved set of cases is called the *adaptation candidates* set. KALES tries to adapt each case in the set for the new machine.

Note that restricting to the machine failure code or the hypothesis is a (domain-dependent) heuristic. The idea behind this is that we try to retrieve from the case memory empirical knowledge *relevant* for diagnosis on the parts for which iMAKE just generated the diagnostic contexts. The experience implicit in a case is described by the symptoms that lead to a fault's recognition. The criterion for case retrieval cannot be the sole appearance of a particular symptom in the case's situation. This would lead to retrieving cases that express the relevance of a machine part for the diagnosis of some other, rather that for the diagnosis of that part itself.

4.3.2. The Second Step : Elaboration of an Adaptation Candidate

Elaborating a candidate means preparing a candidate case for its actual adaptation into the new expert system. This involves mainly producing a justification for the case and analogically transferring the justification to the new machine.

4.3.2.1. Justifying an Adaptation Candidate

Justifying a case means producing an explanation of its situation, that is, of *how* the diagnosed fault was detected and *why* it occurred. Determining how a fault is detected means following the diagnostic situation in the temporal order of symtom gathering and producing in doing this a causal explanation. Detecting why a fault occurred is more complex and we do not intend to provide a conclusive answer to this point now. The cause of a fault might need of the extension of our machine's model with dynamic and geometrical (design) features, which, unfortunately, are still not being modeled.

On the representational level, the explanation of the rule's situation is described by a set of relations between machine parts. The relations can be *structural* (e.g., part $\mathcal{P}_i$ is part of part $\mathcal{P}_k$, or part $\mathcal{P}_i$ is connected to part $\mathcal{P}_k$), *functional* (part $\mathcal{P}_i$'s output is part $\mathcal{P}_k$'s input through port i_k), or *empirical* (denoting that no causal relation—in a strict sense— could be found and that it is the expert who must decide whether a causal relation exists, or is relevant, or none). We record all relations found in the explanation.

On the implementational level, the explanation is a *causal graph* whose nodes are machine parts and whose links are the structural, functional, or empirical relations identified. Identifying the parts involved in the explanation is simple. Each symptom in the situation refers to a machine part (a state) or part's port (an input or output value). For each part referenced by a symptom in the situation, a node is created in the graph.

The creation of the graph follows the order of symptoms in the case's situation. That is,

the graph is first created for the first symptom and the transition to the second one. This is a causal explanation of why a test for the second symptom was chosen. The answer may be, for example, that $\mathcal{P}_2$ is a subpart of $\mathcal{P}_1$, or that a functional relation between $\mathcal{P}_1$ and $\mathcal{P}_2$ is identified in the faulty behavior description of one of those parts, and so on.

The identification of relations between parts in each step of this process follows from a simulation of the (partial) situation on the machine's model and is guided by the context graph. Let $S_1, S_2, \ldots, S_n$ be the diagnostic situation, and $\mathcal{P}_k$ the machine part referenced by symptom S_k. For each pair of symptoms S_k, S_{k+1} in the situation, all possible relations between $\mathcal{P}_k$ and $\mathcal{P}_{k+1}$ are searched, and a link is created between nodes $\mathcal{P}_k$ and $\mathcal{P}_{k+1}$ for each relation found. The link's type is that of the relation. For functional relations, the link is additionally marked with the output value from $\mathcal{P}_k$.

Example:

iMAKE generated the diagnostic contexts for machine part `NewTightenRelease`. Machine part `OldTightenRelease` was found to be similar to the former. This is a part of `OldToolChanger`. Its class is `TightenReleaseDevice`.

Consider the following diagnostic case for `OldTightenRelease`:

```
Name:       #Example1
Situation:  '(and (= FailureCode Code I59)
                  (= Valve Valve5Y1 closed)
                  (= Valve Valve5Y2 open)
                  (= OilConducts OilConductTightenRelease tight)
                  (= Gear AbsorbtionGear clean)
                  (= IoStatus IoStatusIN32 logic-0) )
Diagnosis: EndController5S2atI59Defectuous
```

The causal graph for the situation depicted in this case might look like in figure 4. Once the graph is constructed, KALES proceeds to define an analogical map from the rule's situation to the new machine.

4.3.2.2. Analogical Mapping of the Candidate's Diagnostic Situation

The analogical map of the case's explanation (the causal graph) should lead to producing a new rule (case) for the new system. KALES tries to construct on the new machine a similar explanation graph to that generated in the preceding step.

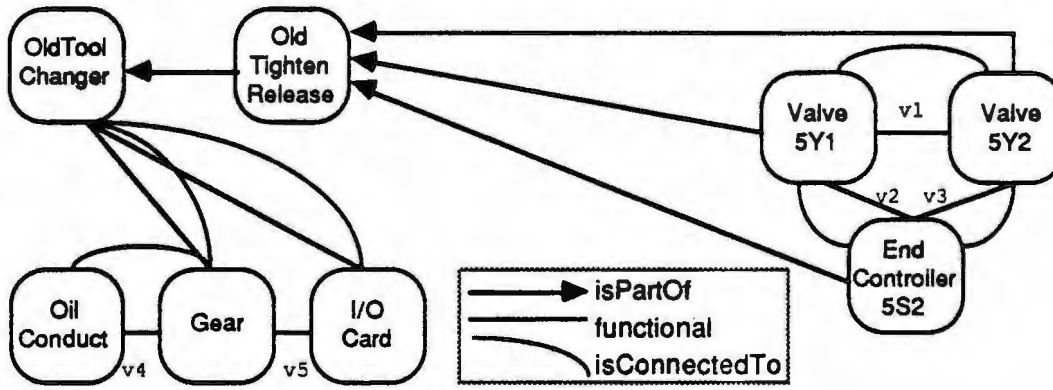
Up to this point, the knowledge adaptation process has been defined independently of the application domain. The same procedure could be applied in other domains provided that they can be modeled and simulated by iMAKE, e.g., other kinds of technical devices. However, completing the analogical process, that is, constructing the situation map, is domain-dependent. To complete the analogical bridge from the old to the new machine we need to search for a graph similar to the explanation graph generated in the preceding step, that can lead to an analogous diagnostic situation.

Deep knowledge from the domain is needed to search for the analogous explanation. We have seen how the machine failure code from the CNC helps constraining the search space to those machine parts referenced by the code. We assume that at least one symptom

in a diagnostic situation contains the failure code received from the CNC. The service technician prepares an association table of machine failure codes of both machines. For instance, if code `159` in the old machine means `ToolChanger`, then that code is associated with, e.g., code `195` corresponding to the tool changer in the new machine.

It is then possible to focus the search for the analogous situation by starting with the first symptom of the new situation being the corresponding machine failure code and its value. The procedure for completing the new situation is an establish-and-refine searching cycle in which the old rule's situation is followed step by step and for each step a new symptom for the new situation is determined.

This determination is guided by the links in the explanation graph. For each symptom processed, the algorithm determines in the new machine's model a corresponding symptom. Finally, a corresponding diagnostic hypothesis must be determined.



— Figure 4 — A Possible Explanation Graph for an Adaptation Candidate —

4.3.3. The Third Step : Incorporation of the New Case in the New Expert System

To complete the generation of the new case, a diagnostic hypothesis must be searched for the produced situation. Furthermore, the hypothesis must then be *tested* before the case can be accepted and incorporated in the new expert system.

To find an hypothesis for the situation, this is simulated through the context graph of the new expert system, by simply letting the expert system search for an appropriate context. This context's name is the new case's solution.

A failure to find an appropriate context may lead to reporting the failure to the service technician, who can either correct the situation, determine the correct context, or reject the situation. A correction of the situation causes KALES to redraw the preceding step's reasoning, by modifying the explanation graph consequently.

The consistent incorporation of the new case in the new system's case memory rests on MOLTKE's Knowledge Base Maintenance and Consistency Checking component (Maurer, 1991). Maurer's component allows the specification of knowledge dependencies on the

definitional level, that is, an object depends on another if its creation, modification, or removal depends on the existence of other objects in the knowledge base.

For knowledge adaptation, dependencies are defined on a different level, namely, on the *justification* of knowledge for its existence. The links on this level, called *justification links*, represent dependencies on a conceptual level. A diagnostic case acquired through adaptation has a justification link poiting to the explanation generated by KALES. The idea behind justification links is to give KALES a means of tracing the knowledge base to the knowledge sources of adapted rules. Thus, knowledge is naturally integrated in a dependency network in which consistency is guaranteed in a TMS-alike fashion[21].

5. STRATEGY ACQUISITION

An *adaptation case* is defined as the procedure through which an old expert system's rule is adapted for the new expert system. It is represented by a triple

$(\mathcal{P}, \mathcal{R}, \mathcal{A})$, where

$\mathcal{P}$ is a pair of parts $(\mathcal{P}.old, \mathcal{P}.new)$, $\mathcal{P}.old$ is an old machine's part and $\mathcal{P}.new$ a new machine's part; $\mathcal{R}$ is a pair of rules $(\mathcal{R}.old, \mathcal{R}.new)$, $\mathcal{R}.old$ is an old expert system's rule and $\mathcal{R}.new$ a new expert system's rule, and $\mathcal{A}$ is the analogy drawn between $\mathcal{R}.old$ and $\mathcal{R}.new$.

During the process of drawing $\mathcal{A}$, paths in the context graph are derived. This means, that for each symptom in $\mathcal{R}.old$, a symptom from the new machine is identified and added to the situation in $\mathcal{R}.new$. This becomes a sequence of symptom determinations, that can be traced on the context graph for the new machine.

KALES abstracts, from a set of adaptation cases relating the same pair of parts $\mathcal{P}.old$ and $\mathcal{P}.new$, and for the same pair of diagnostic hypotheses in $\mathcal{R}.old$ and $\mathcal{R}.new$, an *adaptation strategy*, and associates it to the corresponding diagnostic context.

Adaptation strategies can be compared to shortcut rules in MOLTKE. An empirical shortcut rule determines, with some certainty, some symptom value(s) given a set of known symptoms. An adaptation strategy determines some symptom for the new situation (during analogical transformation) given a set of already determined symptoms. The difference rests on the nature and utility of the determination. Symptom value determination by shortcut rules shortens (thereby improving) the diagnostic process. Determinations reflect the relevance of a certain symptom for detecting a given fault. Symptom determination by adaptation strategies is used to shorten the analogical transformation process. Determinations reflect here the relevance of a certain symptom for building an analogy with another diagnostic situation. An adaptation strategy is a rule that determines the next relevant symptom to include in a situation[22].

---

21 A deeper discussion on the relation between (A)TMS and the knowledge base maintenance component in MOLTKE can be found in (Maurer, 1991)

22 See (Davies and Russel, 1987) for a formal discussion on determinations, and (Dutta, 1988) for an extension of Davies and Russel's approach

6. DISCUSSION

6.1. Causation, Functionality, and Explanation

(Shoham, 1988) points out a very important property of causation, namely, its *context-sensitiveness*. The context in which a cause-effect relation occurs is important for reasoning about the differences and similarities in behavior between two structurally different, functionaly similar mechanisms. Our knowledge adaptation approach allows for expressing differences in structure and behavior between two machines through an analogical transfer of diagnostic knowledge guided by background (causal) knowledge.

However, we must restrict our view of causation to a static analysis. Our formalism still does not allow for dynamical and geometrical modelling of the domain. The analysis of a diagnosis is thus reduced to an isolated sequence of test gathering and contex refinement steps. We analyze causation in the structural and behavioral differences between two machines for isolated machine parts. We traduce *dependency associations* between machine parts features *into causal relationships*[23]. Causal relations in KALES's explanation can be seen as feature correlations of the parts involved in an adaptation case.

Using similarity in functionality is, from our view, an appropriate shortcut for the retrieval of adaptation candidates. In the next future we plan the creation of a deep classification of machine parts. IMAKE classifies parts by the expert-given class name (unfortunatelly, still a very shallow schema) and by its functionality (a part's context subgraph stored with functionality as the differentiating key). We will extend this classification in a deeper structure from which it can be possible to assign knowledge adaptation strategies (seeing this as generalization of the analogical mapping constructed by KALES) to a class or to a class's instance.

6.2. Analogy and Case-Based Reasoning for Improving Adaptation

The analogies drawn by KALES are indeed identification of correspondences on three levels. First, on the machine's structure and behavior: parts retrieved for producing an analogy satisfy some (functional) similarity criteria. Second, on the context graph of both expert systems: an analogous case can be generated only if the analogous situation can be simulated on the context graph for the new machine. Third, on empirical knowledge: the analogy describes how a fault in the new machine can be detected, based on the proceeding on a similar device. The first two levels support (explain) the latter.

Very inspiring work has been that of (Carbonell, 1986) and (Hanson, 1990). Our approach uses a kind of *derivational analogy* through the transfer of decision sequences (test selection) and their justifications (on the causal graph) to build a new diagnostic

23 See, e.g., (Geiger, Paz, and Pearl, 1990) for a discussion on the interpretation of dependencies as causal relationships

under an experimental approach: rules are tested against training case sets generated from available domain data. This is a plausible evaluation strategy for knowledge adaptation. An approach to acquiring strategic knowledge, directed specifically to diagnosis, is described in (Gruber, 1988). Gruber describes an integration of representations to allow for flexible presentation of knowledge to the domain expert and operationalization of the strategic knowledge in the expert system. The aim is giving strategic knowledge a more important role in the knowledge acquisition task. Similarly to our work, Gruber's ASK methodology uses justifications of actions to construct strateic rules.

6.3. Conclusion

Knowledge adaptation by classifying knowledge about structural and behavioral differences and defining a transformation function for rules on the basis of causal dependencies represents an innovative view on (derivational) analogical reasoning. Our approach to knowledge adaptation as a means of knowledge acquisition from past experience intensively exploits background knowledge to justify the acquired knowledge.

ACKNOWLEDGEMENTS

I thank my advisor, Prof. Dr. M.M. Richter, and all my colleagues in the AI Group on Expert Systems at Kaiserslautern, specially Klaus-Dieter Althoff, Frank Maurer, Mike Stadler, Julio Valdés, and Stephan Weß for the fruitful discussions on the topics here presented. The fruits of my work go to my wife, Yalily, for her love and dedication.

BIBLIOGRAPHY

Althoff, K.-D. (1991). Eine fallbasierte Lernkomponente als integrierter Bestandteil der MOLTKE-Werkbank zur Diagnose technischer Systeme, (in english: A Case-Based Learning Component as Integral Part of the MOLTKE Workbench for the Diagnosis of Technical Systems). Doctoral Dissertation, University of Kaiserslautern, FRG, to appear.

Althoff, K-D., De la Ossa, A., Maurer, F., Stadler, M., and Weß, S. (1989). Case-Based Reasoning for Real-World Applications. In: *Proceedings of the FAW Workshop on Adaptive Learning and Neural Networks 1989*, Ulm, Federal Republic of Germany. (Also appeared as SEKI-Report SR-90-22 SFB, Dept. of Computer Science, University of Kaiserslautern, FRG, 1989).

Althoff, K-D., Maurer, F., and Rehbold, R. (1990). Multiple Knowledge Acquisition Strategies in MOLTKE. In: B. Wielinga, J. Boose, B. Gaines, G. Schreiber, M. van Someren (Eds.), *Current Trends in Knowledge Acquisition*, Amsterdam: IOS Press (Proc. of the 4th European Knowledge Acquisiton Workshop, EKAW-90), pp. 21-40.

Althoff, K-D., and Weß, S. (1991a). Case-Based Knowledge Acquisition, Learning, and Problem Solving for Diagnostic Real World Tasks. In: *Proceedings of the fifth European Knowledge Acquisition Workshop,. EKAW-91*, Pre-print (to appear)

Althoff, K.-D., and Weß, S. (1991b). Case-Based Reasoning and Expert Systems Development. In: *Proceedings of the Knowledge Engineering and Cognition Workshop, KECOG*, University of Kaiserslautern, February, 1991 (to appear).

Althoff, K-D., and Trapphöner, R.: GENRULE (1990). Learning of Shortcut-Oriented Diagnostic Problem-Solving in the MOLTKE/3 Workbench. Technical Report, University of Kaiserslautern, Federal Republic of Germany, October 1990.

Bradzil, P.B., and Torgo, L. (1990). Knowledge Acquisition via Knowledge Integration. In: B. Wielinga, J. Boose, B. Gaines, G. Schreiber, M. van Someren (Eds.), *Current Trends in Knowledge Acquisition*, Amsterdam: IOS Press (Proceedings of the 4th European Knowledge Acquisition Workshop, EKAW-90), pp.90-104.

Carbonell, J.G. (1986). Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In: R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning, An Artificial Intelligence Approach*, Vol. II, Los Altos, California: Morgan Kaufmann.

Carbonell, J.G., and Gil, Y. (1990). Learning by Experimentation: The Operator Refinement Method. In: Y. Kodratoff and R.S. Michalski, R.S. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Volume III. Los Altos, California: Morgan Kaufmann, Chap. 7.

Davies, T.R., and Russel, S.J. (1987). A Logical Approach to Reasoning by Analogy. In: J.W. Shavlik and T.G. Dietterich, *Readings in Machine Learning* (1990), San Mateo, California: Morgan Kaufmann, pp.657-663.

Dutta, S. (1988). Learning of Background Knowledge for Making Reliable Analogical Inferences. n: J.S. Gero (Ed.), *Artificial Intelligence in Engineering: Diagnosis and Learning*, Amsterdam: Elsevier, and Southampton: Computational Mechanics Publications, pp. 305-331.

Geiger, D., Paz, A., and Pearl, J.: Learning Causal Trees from Dependence Information. In: *Proceedings of the 1990 AAAI Conference*, pp.770-776.

Gruber, T.R. (1988). Acquiring Strategic Knowledge From Experts. In: J. Boose and B. Gaines (Eds.), *Knowledge-Based Systems Vol. 4: The Foundations of Knowledge Acquisition*, Cambridge: Academic Press, pp.115-133.

Hanson, S.J. (1990). Conceptual Clustering and Categorization: Bridging the Gap between Induction and Causal Models. In: Y. Kodratoff and R.S. Michalski (Eds.), *Machine Learning: An Artificial Intelligence Approach. Volume III.* Los Altos, California: Morgan Kaufmann, Chap. 9, pp.235-268.

Kolodner, J.L. (1983a). Maintaining Organization in a Synamic Long-Term Memory. In: *Cognitive Science*, Vol. 7, pp. 243-280.

Kolodner, J.L. (1983b). Reconstructive Memory: A Computer Model. In: *Cognitive Science*, Vol. 7, pp. 281-328.

Kolodner, J.L. (1989). The Mediator: Analysis of an Early Case-Based Problem Solver. In: *Cognitive Science*, Vol. 13, pp. 507-549.

Maurer, F. (1991). Knowledge Base Maintenance and Consistency Checking in MOLTKE. In: *Proceedings of the Sixth Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW-91*, Banff, Canada, October 1991, to appear.

Mitchell, T.M., Utgoff, P.E., and Banerji, R.B.: Learning by Experimentation: Acquiring and Refining Problem Solving Heuristics. In: R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning, An Artificial Intelligence Approach*, Berlin: Springer-Verlag, pp. 163-190.

Morik, K. (1987). Sloppy Modeling. In: K. Morik (Ed.), *Knowledge Representation and Organization in Machine Learning*, Lecture Notes in Artificial Intelligence 347, Berlin: Springer-Verlag, pp.107-134.

Morik, K. (1988). Acquiring Domain Models. In: J. Boose and B. Gaines (Eds.), *Knowledge-Based Systems Vol. 4: The Foundations of Knowledge Acquisition*, Cambridge: Academic Press, pp. 245-256.

Murray, K.S. (1991). Learning as Knowledge Integration. Ph.D. Dissertation Extended Abstract, Dept. of Computer Sciences, University of Texas at Austin, February 1991.

Rehbold, R. (1991). Die Integration von modellbasiertem Wissen in technische Diagnose-expertensystemen (in english: The Integration of Model-Based Knowledge in Technical Diagnosis Expert Systems). Doctoral Dissertation, University of Kaiserslautern, FRG, 1991.

Schreiber, A.Th., Wielinga, B.J., and Breuker, J.A. (1991). The KADS Framework for Modelling Expertise. In: *Proceedings of the fifth European Knowledge Acquisition Workshop,. EKAW-91*, Pre-print (to appear)

Shen, W.M., and Simon, H.A. (1989). Rule Creation and Rule Learning Through Environmental Exploration. In: *Proceedings of the International Joint Conferenfe on Artificial Intelligence 1989*, Morgan Kaufmann.

Shoham, Y. (1988). Reasoning About Change. Time and Causation from the Standpoint of Artificial Intelligence. Cambridge: MIT Press.

Struss, P. (1988). Extensions to ATMS-Based Diagnosis. In: J.S. Gero (Ed.), *Artificial Intelligence in Engineering: Diagnosis and Learning*, Amsterdam: Elsevier, and Southampton: Computational Mechanics Publications, pp. 3-28.

A Multiple-Method Knowledge-Acquisition Shell for the Automatic Generation of Knowledge-Acquisition Tools

Angel Puerta, John Egar, Samson Tu, and Mark Musen
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, CA, 94305-5479
puerta@camis.stanford.edu

Abstract

The use of predefined models of problem-solving methods is receiving considerable attention from researchers in the area of knowledge acquisition. Using these models, developers of knowledge-acquisition tools are able to prescribe the roles in which knowledge is used in completing a given task. A number of *method-oriented* architectures based on a single problem-solving method have been developed by various research groups. Because the methods are domain-independent, method-oriented architectures are limited by the fact that knowledge roles that depend on domain-specific considerations cannot be represented using the model of problem solving. In addition, the interface between the knowledge-acquisition tool and the application expert cannot adequately convey the role of each knowledge type in the task model. PROTÉGÉ-II is a knowledge-acquisition shell that we are building to generate knowledge-acquisition tools automatically without presupposing a specific model of problem-solving. The shell manages a library of mechanisms—procedures of grain size smaller than that of problem-solving methods. Mechanisms can be combined in PROTÉGÉ-II to construct problem-solving methods and to define the roles of knowledge that depend on domain considerations. Furthermore, PROTÉGÉ-II utilizes the concept of *adaptation* in interfaces to allow the knowledge engineer to produce interfaces that are task- and domain-specific. In this paper, we present the PROTÉGÉ-II shell and examine the components of its architecture. We also demonstrate the use of PROTÉGÉ-II with a running example, and discuss the design techniques used to overcome the limitations of method-specific architectures.

1. Method-Oriented Architectures

A recent focus of research in knowledge acquisition is the use of models of domain-independent problem-solving methods to construct knowledge-acquisition tools [McDermott, 1988]. These models of problem solving, such as heuristic-classification [Clancey, 1985] and skeletal-plan refinement [Friedland and Iwasaki, 1985], allow a knowledge engineer to develop a model of the task area at hand in terms of the abstract problem-solving method. Thus, task- and domain-specific knowledge-acquisition tools can be obtained from a task- and domain-independent model. Since the problem-solving models are independent of any knowledge-representation formalism, the modeling of tasks occurs at the *knowledge level* [Newell, 1982], where only the role of each type of knowledge is specified, as opposed to at the symbol level, where the representation of each type of knowledge must be described. There are several examples of knowledge-acquisition tools built from models of problem solving. ROGET [Bennett, 1985] used a specialized form of the heuristic-classification method to acquire knowledge for diagnostic tasks. SALT [Marcus and McDermott, 1989] used a *propose-and-revise* method for configuration tasks.

There is also another category comprising tools that operate at a metalevel. These tools are able to generate knowledge-acquisition tools automatically from a model of a task. Examples of this type of tool are PROTÉGÉ [Musen, 1989a; Musen, 1989b], which is method-oriented, and DOTS [Eriksson, 1990], which does not follow a given problem-solving method.

PROTÉGÉ, which was developed in our laboratory, views the problem of automating the construction of knowledge-acquisition tools as one of generating an interface to a knowledge editor that is utilized by an application

expert. The components of the interface are determined by a task model developed from the method of skeletal-plan refinement [Musen and Tu, 1991]. PROTÉGÉ makes extensive use of graphical interaction modalities, such as graphical editors, which are especially well suited for capturing procedural knowledge (e.g., a flowchart diagram). Because the interfaces are based on the task model for the domain of interest, PROTÉGÉ-generated tools can guide the user through the acquisition sessions, ensuring that the knowledge captured is complete and consistent with respect to that model [Musen et al., 1987].

Like other method-oriented knowledge-acquisition tools, PROTÉGÉ suffers because its domain-independent problem-solving method cannot define the role of domain-dependent control knowledge. For example, in the domain of cancer therapy, there is no way for the knowledge engineer to use the skeletal-refinement method to declare how the effects of various specifications for altering the dose of a drug should be combined, because the concept of a *dose adjustment* is a domain-specific one. In these instances, a knowledge engineer must fall from the knowledge level to the symbol level to enter the required knowledge in a particular representation (e.g., by specifying a production rule or by changing the ordering of rules).

Our current research aim is to identify building blocks, called *mechanisms*, of a grain size finer than that of problem-solving methods, that can be (1) combined to construct problem-solving methods, and (2) applied to define the role of domain-dependent knowledge [Tu et al., 1991]. Our approach is essentially empirical in that we hope to identify several of these mechanisms from the problem-solving methods being applied in our current development of various medical expert systems. After the mechanisms are identified, we will test their applicability to other tasks and domains, and their combinability into problem-solving methods different from the ones from which they were extracted.

Other groups are carrying out parallel research to find ways to define the roles of domain-dependent and domain-independent knowledge at the knowledge level. Examples are Chandrasekaran's research on generic tasks [Chandrasekaran, 1986], Steels' work on the componential framework [Steels, 1990], and McDermott's study of role-limiting methods [McDermott, 1988].

1.1. A New PROTÉGÉ

The use in our group of mechanisms for the generation of knowledge-acquisition tools has dictated the need for reimplementation of PROTÉGÉ. To address this need, we are developing PROTÉGÉ-II, a knowledge-acquisition shell that provides a task-modeling environment for knowledge engineers and a knowledge-editing environment for application experts. The shell is independent of methods, tasks, and domains. PROTÉGÉ-II will allow the definition of knowledge roles at the knowledge level for both domain-dependent and domain-independent knowledge. This shell will permit the construction of problem-solving methods using mechanisms as building blocks, the modeling of application tasks in terms of the constructed methods, the generation of knowledge editors based on those task models, and the acquisition of knowledge from such knowledge editors.

In creating PROTÉGÉ-II, we have to remove two fundamental types of limitations in PROTÉGÉ: limitations produced by presupposition of a problem-solving method, and limitations produced by presupposition of an interface style. We have mentioned the limitations caused by the single-method problem. The limitations imposed by the interface style, however, may be just as crucial. PROTÉGÉ has no flexibility in presenting the contents of a knowledge base to the application expert. The presentation style is fixed, and is based on the model of skeletal plan-refinement, which is domain-independent. Therefore, regardless of the task or domain, similar types of knowledge are always presented in the same style to the application expert. The lack of flexibility in the style of the interface in PROTÉGÉ is due to the absence of adaptation capabilities in the generated interfaces. Adaptation is the process of varying the contents and behavior of an interface based on the characteristics of the interaction. Whereas adaptation in interfaces is usually related to the characteristics of the user, the features of the task and domain of the interaction are relevant as well [Puerta, 1990; Rissland, 1984]. To be able to generate interfaces to knowledge editors without knowing a priori the problem-solving method used for task modeling, PROTÉGÉ-II must provide the means for the generated interfaces to adapt to the task and domain of interest.

In this paper, we present the implementation of the method-independent PROTÉGÉ-II shell. In describing the system, we show the techniques applied to overcome the limitations of the original version of PROTÉGÉ. In particular, we show how the construction of problem-solving methods from mechanisms can be achieved using a librarylike storage–retrieval system and a graphical editing environment. We also describe the use of two formal languages developed to

provide adaptability in the interfaces generated by PROTÉGÉ-II. We claim that these languages provide independence from task and domain considerations in the interface-generation process, and give the knowledge engineer freedom to select specific presentation styles for each type of knowledge to be captured in the knowledge editors. Through the use of a user-interface management system (UIMS), PROTÉGÉ-II provides consistency in the interaction with the user throughout the various phases of knowledge-editor generation, and coordinates the complex relationships among interface components. Through the use of PROTÉGÉ-II, we hope to show how knowledge-acquisition tools can benefit from having access to multiple problem-solving methods and from presenting task-specific and domain-specific interfaces to the application expert.

The remainder of this paper is organized as follows. Section 2 introduces terminology necessary for the rest of the discussion. Section 3 offers an overview of the PROTÉGÉ-II system, describing the various subsystems in the shell and their role in the process of knowledge-editor generation. Section 4 presents an example of the use of PROTÉGÉ-II. Note that, since PROTÉGÉ-II is in a prototype stage, only part of the functionality described in Section 3 will be demonstrated in Section 4. To conclude, Section 5 discusses several research issues associated with the development and use of PROTÉGÉ-II.

2. Key Concepts

It is important that the concepts associated with the words *task* and *method* in PROTÉGÉ-II be specified clearly. In this section, we present these concepts along with definitions for the building blocks of the knowledge-level model of PROTÉGÉ-II (problem-solving mechanisms), and the two basic operations on these building blocks allowed by PROTÉGÉ-II: method configuration and method assembly. Given that the focus of this paper is on the architectural concept of PROTÉGÉ-II and its related human–computer interaction issues, our description of problem-solving mechanisms is intended to merely anchor the subsequent discussions. A more complete presentation of our views on methods and mechanisms has been presented elsewhere [Musen and Tu, 1991; Tu et al., 1991].

2.1. Tasks

A *task* in PROTÉGÉ-II is an activity, or an abstraction of an activity, in the real world. A task accepts some type of input and produces some type of output. The domain to which the task is applied determines the type of inputs accepted and the type of outputs produced.

Just as important as saying what a task is, is defining what a task is not. For our purposes, a task by itself is not a composite structure. Thus, it does not support a view of subtasks, or a decomposition into a hierarchy of subtasks. Such decompositions are possible, but only in the context of a problem-solving method, as shown is Section 2.3. In addition, a task does not impose a requirement on the type of knowledge needed to complete that task. This perspective is in contrast with the view of other researchers [Vanwelkenhuysen and Rademakers, 1990], but allows us to specify cleanly the functionality of PROTÉGÉ-II based on the manipulation of mechanisms and methods.

2.2. Mechanisms

A *mechanism* in our library is a procedure that completes, or solves, a task. It is a specification of *how* a task is accomplished. There is a many-to-many relationship between tasks and mechanisms that will serve as a base for defining the processes of method configuration and method assembly. The finite set of tasks that can be completed using a given mechanism is defined as that mechanism's *target;* the finite set of mechanisms that can complete a task is referred to as that task's *source.*

A mechanism imposes on its target tasks various requirements that determine what types of knowledge and data must be available to solve the task with that mechanism. In our implementation, these requirements derive from the components that make up a mechanism. These components are as follows [Musen and Tu, 1991]:

1. An input–output (I/O) declaration: The input declaration stipulates what inputs are required by the mechanism. The output declaration determines the target tasks of the mechanism.

2. A global data model: This component specifies the type of input data that the mechanism accepts and the type of output it produces, and the classes of operations that can be performed on the data.

3. A set of semantic constraints: This set defines relationships between inputs and outputs; thus, knowledge to verify the constraints must be accessible in the task.

4. A control- and data-flow configuration: The particular configuration used by a mechanism dictates that certain types of knowledge about the task be readily obtainable to allow the defined flow to operate. In other words, decisions that alter the flow of data or control may be dependent on certain knowledge about the task.

2.3. Methods

The union of the target tasks of all mechanisms in the PROTÉGÉ-II library neither is a complete set of all possible tasks, nor is intended to be one. There will be tasks that will fall outside the target of any of the existing mechanisms. It is possible, nevertheless, to assemble two or more mechanisms into a *method*. Mechanisms and methods hold the same many-to-many relationship with tasks. The target tasks of methods, however, are different in nature from the target tasks of mechanisms. Figure 1 illustrates this situation. So that we can differentiate the two types of tasks distinguishable when tasks are linked to methods or mechanisms, we call those that are part of a mechanism's target *simple* tasks, and call those belonging to a method's target *composite* tasks. A method in our library is a procedure that decomposes a composite task into subtasks, some of which may be composite as well. The decomposition process continues until all composite tasks and subtasks are reduced to simple subtasks. At this point, mechanisms can be applied to complete, or solve, the simple subtasks and, consequently, the original composite task. In addition to the requirements imposed on the tasks by the mechanisms making up a method, there may be requirements imposed on the composite task by the method itself. These requirements result from the assembly of the mechanisms into a procedure—the method—that has an overall data- and control-flow configuration.

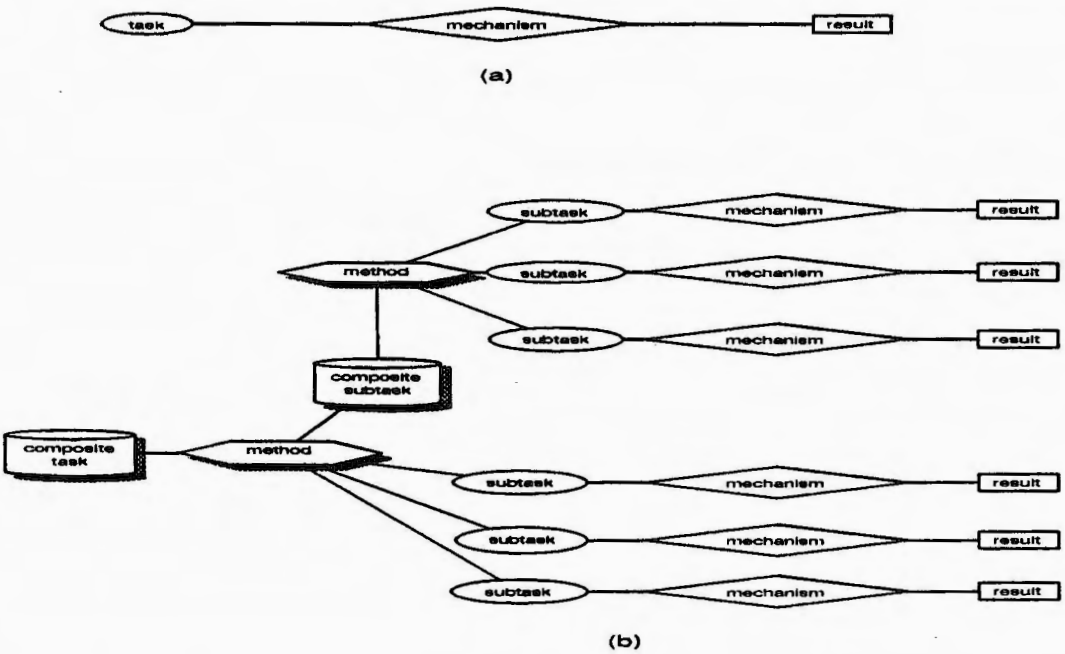


Figure 1. Methods, mechanisms, and tasks. In (a), a task is considered *simple* when it can be completed with a mechanism. In (b), a task is *composite* (noted by the shadowed outline) when it must be decomposed, perhaps recursively, into simple tasks, as imposed by a method, before a result can be obtained. A subtask is just a task that forms part of a task decomposition.

2.4. Method Configuration

The process of determining which mechanism, or method, should solve each subtask in a task decomposition is called *method configuration*. As shown in Figure 2, method configuration connects a task with a mechanism, or with a method, in the task's source. It is only when this connection is made that the task becomes composite or simple. If the task is connected to a mechanism, then it is considered *simple*. If it is connected to a method, then it is viewed as *composite* and must be broken into subtasks. After task decomposition is achieved, we complete the configuration process by configuring single mechanisms for each of the resulting simple subtasks.

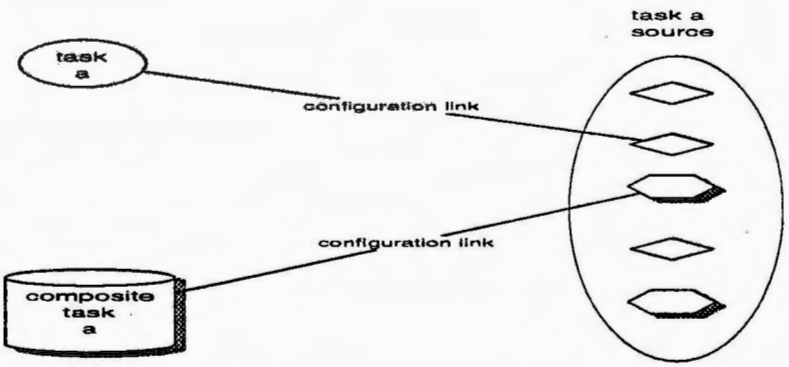


Figure 2. Method configuration. Configuring a method consists of establishing a link between a task and a mechanism, or a method, in the task's source. The same task can be viewed by PROTÉGÉ-II as simple or composite, depending on the link.

2.5. Method Assembly

The process of defining a new target by combining two or more mechanisms into a method is called *method assembly* and is illustrated in Figure 3. Putting together several mechanisms defines a task decomposition, which in turn determines a target of composite tasks fitting such decomposition.

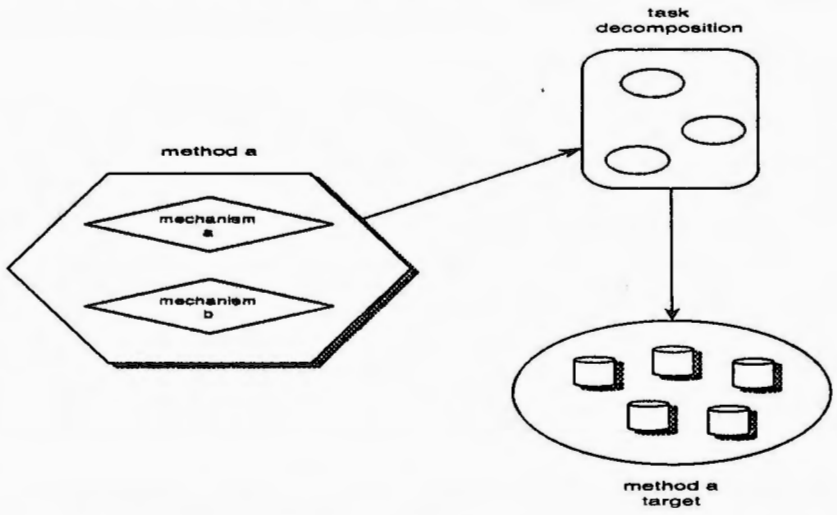


Figure 3. Method assembly. When a method is assembled through a combination of two or more mechanisms, a task decomposition for the method results. The task decomposition specifies how a composite task must be decomposed into subtasks so that the method can be used to solve the composite task. This task decomposition also defines a set of target composite tasks that can be decomposed in the manner dictated by the decomposition.

Note that the task decomposition resulting from the method assembly is what permits method configuration on the assembled method to take place at a later time. Although a method is assembled with individual mechanisms, PROTÉGÉ-II views each method by only that method's task decomposition. Therefore, at method-configuration time, PROTÉGÉ-II looks for the mechanisms that can complete the subtasks in the method's task decomposition. These mechanisms include, but are not limited to, the ones with which the method was assembled originally.

3. The PROTÉGÉ-II Shell

PROTÉGÉ-II is a general-purpose knowledge-acquisition environment that assists knowledge engineers in creating specialized knowledge-acquisition tools (knowledge editors) that are used by application experts to enter knowledge

into a knowledge base. Thus, PROTÉGÉ-II is considered a *metatool*. Unlike the original implementation of PROTÉGÉ, the reimplementation described in this paper does not require users to define knowledge in terms of a presupposed problem-solving method. Instead, users are given the ability to configure or assemble methods that will solve satisfactorily the class of tasks at hand.

Figure 4 shows an overview of the architecture of PROTÉGÉ-II. Knowledge engineers use the system to configure or assemble a method and, eventually, to develop a model of the task area for which knowledge is to be acquired. The system generates automatically a knowledge editor, which interacts with application experts to capture task-specific knowledge. A method- and domain-independent advice system uses the contents of the knowledge base and the given method to produce responses to its end users.

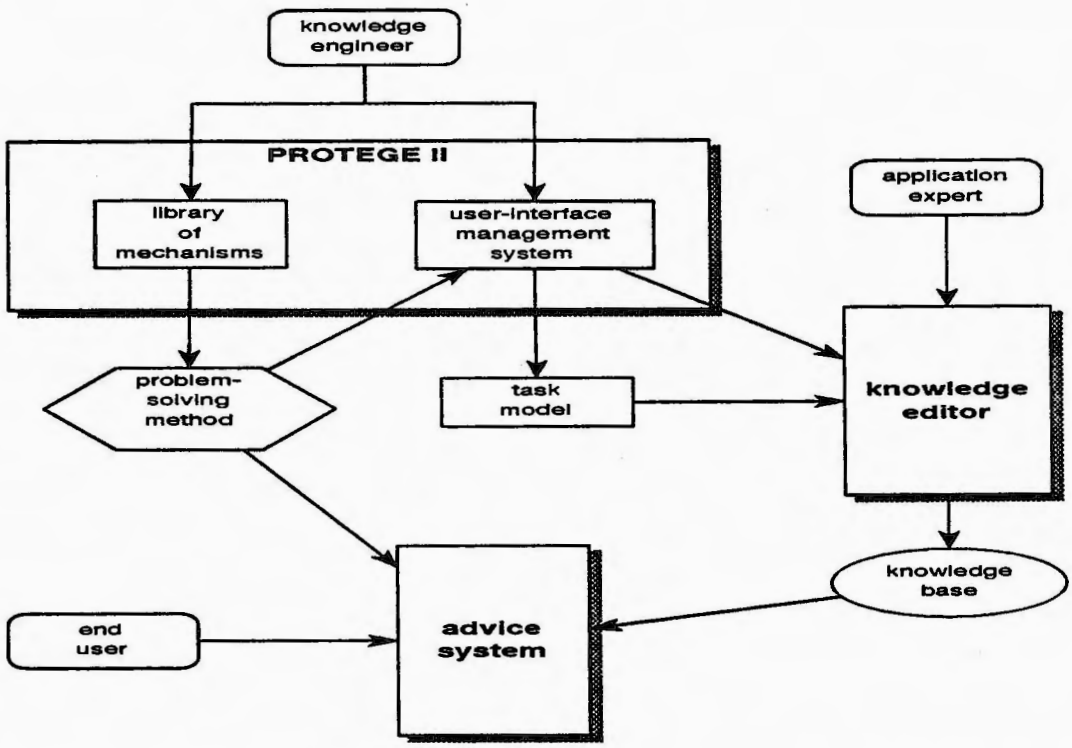


Figure 4. The PROTÉGÉ-II knowledge-acquisition shell. A knowledge engineer uses the facilities of the library of mechanisms to construct a problem-solving method. This method dictates how a model of the task is developed. Based on the task model, the knowledge engineer uses the user-interface management system to generate a knowledge editor. The application expert enters knowledge into a knowledge base using the editor. The knowledge base is ultimately utilized by an advice system that solves problems according to the constructed problem-solving method.

By providing configuring and assembling capabilities, PROTÉGÉ-II allows knowledge engineers to custom-tailor problem-solving methods for specific tasks. Furthermore, the resulting methods will have sufficient semantics for users to define at the knowledge level the roles that the entered knowledge will play in problem solving [Musen and Tu, 1991]. The previous PROTÉGÉ implementation forced users to define many of these roles at the symbol level.

3.1. General System Architecture

The architecture of PROTÉGÉ-II has two main subsystems (see Figure 4). Method assembly and configuration are conducted using the library of mechanisms. Task modeling and knowledge-editor generation are accomplished through the UIMS. This latter subsystem also defines the appearance and behavior of the generated interfaces, and the control of their various interface components. In Sections 3.2 through 3.4 these subsystems are described in further detail. The library of mechanisms was designed to overcome the limitations imposed in PROTÉGÉ by the assumption of a singular

problem-solving method. The UIMS was created to provide a high degree of freedom in the specification of the interaction style of the generated interfaces.

3.2. The Library of Mechanisms

The main function of the library of mechanisms is to provide the knowledge engineer with the facilities to look up mechanisms and methods, and to configure, or assemble, these methods in order to custom-tailor a problem-solving method for the task area at hand. In Figure 5, this subsystem has been further decomposed. As would be expected in any library, the index system is the central component. The index provides the means for organizing the search and selection of mechanisms. Depending on the mode (configuration or assembly) that the user selects, the index system presents either a configuration palette or a graphical assembly editor, respectively, to the knowledge engineer. The palette and the editor allow the necessary tailoring of candidate methods to produce the final problem-solving method. Note that the resulting method is, at this point, still domain-independent. The knowledge engineer will specialize this method for the particular domain during the phase of knowledge-editor generation.

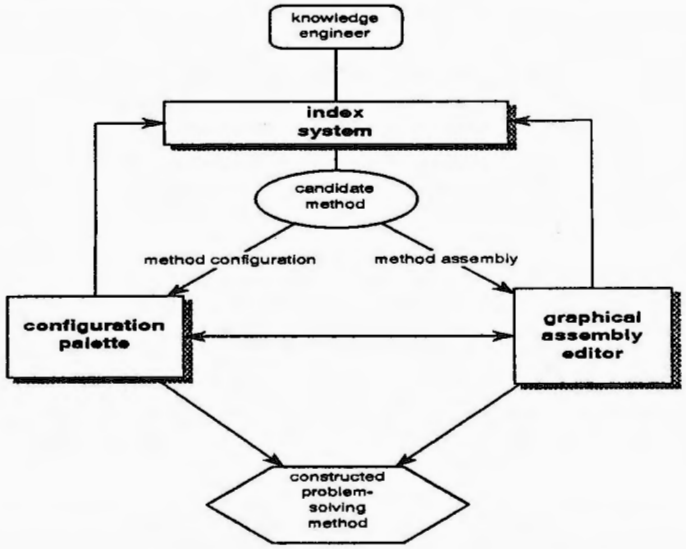


Figure 5. The library of mechanisms. Knowledge engineers search for candidate methods using the index system, which provides multiple search strategies. Candidate methods can be configured or assembled to obtain a final method that fits the task at hand. Typically, the process will require several iterations among the index system, the configuration palette, and the graphical assembly editor.

The underlying philosophy in the implementation of the index system is to give power to the knowledge engineer. We are not ready to remove the knowledge engineer from the cycle of expert-system construction. Instead, we want to present information to this user in a way that takes maximum advantage of the engineer's skills for selecting a suitable problem-solving method. This job can be equated with a student's preparation of a report on a given subject. The student may employ book-library facilities to find an already-written report on the subject, and may change, or supplement, some sections in that report using additional sources (giving proper credit, of course!). This action is the equivalent of method configuration. The student could also construct a new report by combining material from several sources in a process similar to method assembly. The library of mechanisms in PROTÉGÉ-II assumes that the knowledge engineer has the ability to judge the applicability and compatibility of methods and mechanisms, and has the skills to construct a problem-solving method using the available building blocks. In the same way that a library cannot prepare a report for a student, PROTÉGÉ-II will not, at the touch of a button, prepare a solution for the knowledge engineer's problem.

Indexing of mechanisms in PROTÉGÉ-II is based on the components of each mechanism and on its target tasks. There are three indexing strategies that we have identified as useful: by task, by global data model, and by I/O declaration.

The strategies complement one another, and use of one for a portion of the method configuration or assembly process does not preclude use of any other as a supplement.

3.2.1. Indexing by Task

Each mechanism, or method, is associated in the library with a set of tasks, or with a set of composite tasks, respectively. There are two ways in which this association can take place. First, when the mechanism is initially incorporated into the library, tasks can be specified to be associated with the mechanism. Second, during the normal use of the library, the knowledge engineer can establish that a mechanism is applicable for a task, and this information can be added to the library. When a search by task is conducted and a given task is selected, the index system lists all the mechanisms and methods associated with the task. Notice that this indexing scheme does not define a hierarchy of tasks.

Browsing through a list of tasks in the library of mechanisms has implicit limitations. One shortcoming is that the task's name as given by PROTÉGÉ-II, or the textual description for that name, may not exactly match the name and description that the PROTÉGÉ-II user is trying to find. In PROTÉGÉ-II, perfect matching is neither sought nor assumed likely. Instead, the general notion of custom-tailoring is applied, and the user is expected to navigate the process of constructing the problem-solving method. Consequently, the multiple-indexing capabilities of the library are of importance to facilitate the construction operation.

3.2.2. Indexing by Global Data Model

A mechanism specifies the type of data structures that it can accept through its data model. The model is global in the sense that it provides the base types with which PROTÉGÉ-II users can define concepts in the task's domain and build a task model from which a knowledge editor can be generated. The data model imposes limitations on the applicability of a given mechanism; thus, the data model can be used as part of the search criteria.

The index system can display a list of known global data models, and, on selection of one of these, can display the mechanisms and methods that use the data model. As with task indexing, no hierarchy of data models is constructed.

3.2.3. Indexing by Input–Output Declaration

The inputs and outputs that a mechanism requires also constitute a limiting factor in mechanism applicability. A mechanism, even though it may belong to a particular task's source, may not be able to complete that task in a given instance because all the inputs that it needs are not available. Furthermore, its outputs may not be as complete as desired, or may be insufficient to be assembled with another mechanism. The listing of inputs or outputs in browsers by the index system is similar to that of the other indexing strategies.

This strategy, when combined with the other two, affords the knowledge engineer multiple possibilities to search for candidate mechanisms and methods. This ability is important because we cannot determine a priori which indexing scheme constitutes the best limiting factor in a mechanism search. If, after employing the index system, the choice is a method, the user can configure, or assemble, this method using a configuration palette and an assembly editor.

3.2.4. The Palette and the Editor

The configuration palette and the graphical assembly editor are the tools that the knowledge engineer applies to construct a final problem-solving method that is custom-tailored for the given task. The graphical assembly editor allows the user to manipulate directly graphical objects to assemble new methods. It is more flexible than the palette, which simply permits the substitution of one of the method components for another. There is no restriction on using both tools simultaneously to, for example, configure a method by assembling a new method to fit as one of the configured method's components.

Our description of these subsystems is included with the discussion on the library of mechanisms because these functions are relevant in this context. The actual display and management of the subsystems' interfaces is conducted by the UIMS. This major subsystem is described in Sections 3.3 and 3.4.

3.3. The User-Interface Management System

Once a problem-solving method has been tailored, the knowledge engineer has to model the task area under the terms dictated by the method. Based on the task model, PROTÉGÉ-II then generates a knowledge editor whereby the

application expert can enter knowledge into the knowledge base. Both task modeling and knowledge editing are highly interactive processes in PROTÉGÉ-II. The interactive nature of the system places great demands in the definition and specification of the interfaces used by the knowledge engineer and by the application expert. This circumstance, coupled with the stated requirement of providing flexible interaction styles and the fact that the knowledge-editor interfaces must be generated automatically, creates a complex environment for interface specification. So that we can control this environment, we are developing a UIMS that permits a straightforward selection of interaction styles for different types of knowledge, and that directs the generation and display of these interfaces. Using the functionality of the UIMS, a knowledge engineer can *adapt* an interface to the characteristics of a task, a domain, and a method.

Interactions between PROTÉGÉ-II and knowledge-editor users take place through the use of forms or through the manipulation of graphical objects in an editor (Figure 6). The forms and graphical-objects compilers take programmatic textual descriptions written in a formal language, and produce specialized interfaces that guide the knowledge-acquisition routine by enforcing the domain-specific constraints specified through this formal language. The output of the forms compiler is a hierarchy of forms; the output of the graphical-objects compiler is an editor palette composed of graphical objects that can be combined in the editor according to a given editing paradigm (e.g., nodes-and-links, jigsaw puzzle). The current implementation of PROTÉGÉ-II supports only nodes-and-links graphs.

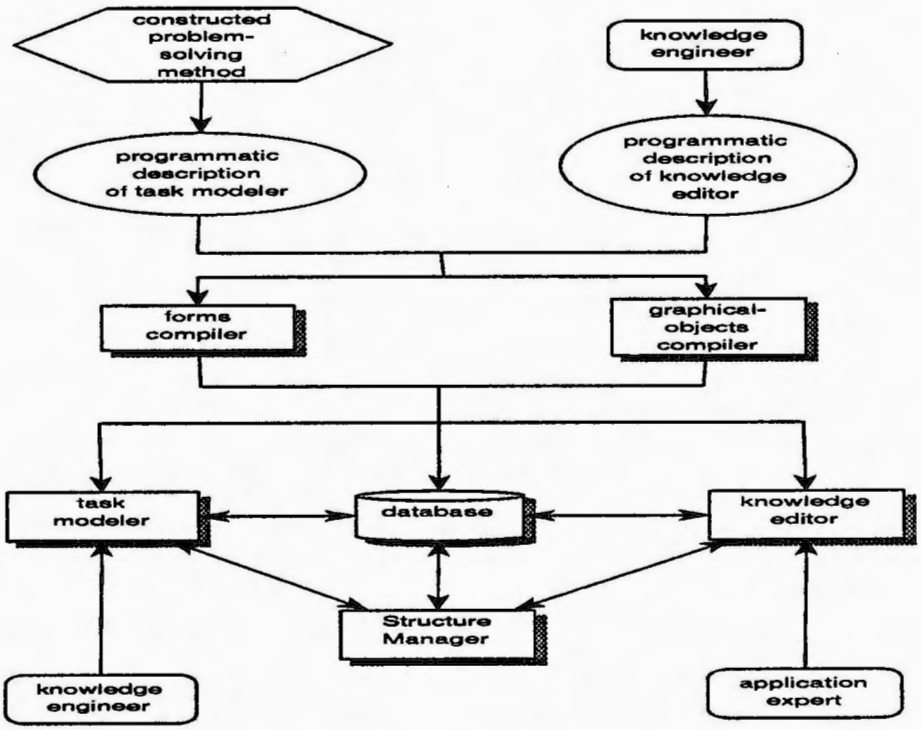


Figure 6. The user-interface management system. Programmatic descriptions of the task modeler and knowledge editor are processed by compilers that generate the necessary interfaces. A structure manager coordinates the display of all interface components using information stored in a common database.

Associated by design with each problem-solving method are programmatic descriptions of forms and of graphical objects that determine how the knowledge engineer can model the current task according to the chosen method, and how the domain-specific terms in the task can be defined. The association is made at method-entry time in the library of mechanisms, and the textual descriptions are a necessary part of each method's definition. Links between graphical objects and forms are maintained through the structure manager of the UIMS. This subsystem coordinates the interdependencies among forms and graphical objects, and manages the display of all structures according to these dependencies during task-modeling and knowledge-editing sessions.

To generate a new knowledge editor, the knowledge engineer makes use of the structures (forms and graphical objects) associated with the problem-solving method to enter the domain-specific terminology needed to model the task. These structures are presented to the knowledge engineer in the task modeler (see Figure 6). After building a model of the task, the knowledge engineer selects interaction styles for the various types of knowledge (e.g., nodes-and-links graphs for procedural knowledge) and PROTÉGÉ-II generates a *preliminary* knowledge editor based on those selections. Since this knowledge editor has been generated from a problem-solving method that is domain-independent, the knowledge editor is not yet adapted to the needs and requirements of the user (the domain expert), or to the particular task and domain. The adaptation of the knowledge editor is accomplished by the knowledge engineer who customizing the visual appearance of the graphical objects, and who specifies constraints, using formal languages, that will guide the knowledge-acquisition process. Finally, the knowledge engineer employs the facilities of the structure manager to stipulate the display relationships among forms and graphical objects. Using the structure manager, the knowledge engineer ultimately specifies how the inputs required by the problem-solving method relate to the knowledge entered in the knowledge editor. The editors generated by PROTÉGÉ-II are similar to that of OPAL [Musen et al., 1987]. There is an implicit assumption that the use of formal languages to generate interfaces provides enough flexibility to accommodate the varying requirements of the methods in the library. In Sections 3.3.1 through 3.3.2 we take a closer look at what is involved in the knowledge engineer's job of editing programmatic descriptions for the forms and graphical-objects compilers, and at how these descriptions shape the generation of the knowledge editors. The features described in these sections are exemplified in Section 4, where the use of PROTÉGÉ-II is demonstrated.

3.3.1. The Forms Compiler

The form-based interfaces used by the application expert, and those associated with problem-solving methods, are defined in a programming language, called FormIKA [Bennett, 1990], which was developed specifically for PROTÉGÉ-II. A program in this language has three goals:

1. Definition of variable-sized forms with multiple columns, value-entry fields, and value-selecting buttons
2. Definition of relationships among forms by creation of a hierarchy of forms in the interface
3. Definition of relationships among entries in the forms constituting a constraint system for the interface

An important part of the FormIKA language is the constraint-specification constructs. Whereas many user interfaces maintain relationships among interface objects by applying procedural attachments, FormIKA defines such relationships declaratively. Thus, knowledge engineers can use this language to declare domain-specific constraints that apply to particular pieces of knowledge that are acquired through the hierarchy of forms. In this manner, the constraints are used to guide the knowledge-acquisition session with the application expert and to adapt the knowledge-acquisition tool to the particular task, domain, and user. The same language is used by PROTÉGÉ-II to generate forms for the preliminary knowledge editor. These preliminary forms will have some constraints already specified that are generated from the model of the task constructed in the task modeler.

Constraints in FormIKA can be visual or nonvisual. Visual constraints deal with the appearance of entry blanks in the forms. Based on the value entered by the application expert in a particular entry blank, the constraint system can make other entry blanks accessible (visible), or inaccessible (nonvisible), thus directing the attention of the expert to the appropriate blanks. Nonvisual constraints work in a similar way, but affect instead the values displayed on the blanks themselves. Also, the constraint system can update or determine values in some entry blanks by applying defined constraints that relate to other entry blanks whose values have been entered already. In this manner, the consistency of the acquired knowledge is maintained.

3.3.2. The Graphical-objects Compiler

Both task modeling and knowledge editing can take place in graphical editors through manipulation of graphical objects (e.g., drawing a flowchart). These objects and their behavior can be specified through a formal language called Palette Constraint Language (PCL). Programs in this language have the following purposes:

1. Definition of a number of graphical objects that constitute a *palette* for a graphical editor
2. Definition of internal constraints that determine the behavior of the objects on the screen
3. Definition of external constraints that determine the connectivity of the graphical objects when these are used in a graphical editor

Again, the purpose of this language is to permit the definition of the constraints in a declarative fashion. The constraints have the effect of adding domain-specific behavior to otherwise general graphical editing structures, thus steering the interaction with the application expert.

Internal constraints limit the editing functions applicable to particular objects. For example, some objects can be declared as modifiable, giving the knowledge-editor user the ability to define subtypes of that particular object that inherit the parent's behavior. This feature is useful in instances where the task model defined by the PROTÉGÉ-II user is incomplete or is not sufficiently specialized, requiring the expert to define new terms derived from the more general task model.

External constraints put restrictions on the ability to connect one object to another. This type of constraints also is domain-dependent, and, together with the internal constraints, defines a behavior for the knowledge editor that will be generated. As noted before, PROTÉGÉ-II currently generates editors with a nodes-and-links editing paradigm. We are generalizing the grammars of FormIKA and PCL to form a single language that can support multiple views (forms, graphs) and multiple editing paradigms.

3.3.3. The structure manager

The on-screen display and handling of forms and graphical objects is accomplished with the structure manager. This subsystem can be used to bring up a specific interface structure, such as a form or a graphical object, or to associate one structure with another. The PROTÉGÉ-II user can relate the display of a particular form dynamically by double-clicking on an individual graphical object. The user can also associate an entry blank in a form with an object, or the label of a graphical object with a form. The structure manager also serves the vital role of linking the library of mechanisms with the task modeler and the knowledge editor. The special implications of this link are detailed in Section 3.4.

3.4. Communication Among Subsystems

Up to this point, we have described the library of mechanisms and the UIMS as two completely separate entities. The generation of a knowledge editor has been represented as a step-by-step process that moves from one subsystem to the next in an orderly manner. It is also possible, however, to access the library of mechanisms in PROTÉGÉ-II during the phases of task-modeling and knowledge-editor generation.

This link enables the knowledge engineer to define at the knowledge level the roles played by domain-specific control knowledge. For example, in the domain of hypertension, the structure manager can make the necessary links so that a drug dosage can be calculated using a specific mechanism (or set of mechanisms) from the library. Relationships between graphical objects and the library can be prescribed in similar fashion.

By means of this functionality, the PROTÉGÉ-II user can ultimately connect particular inputs of a method to forms, entry blanks, or graphical objects in the knowledge editor or in the task modeler. The definition of the roles of pieces of knowledge at the knowledge level was not always possible with the previous implementation of PROTÉGÉ [Musen and Tu, 1991]. In that system, many of these roles had to be defined at the symbol level by the knowledge engineer either as production rules or as procedures. By not presupposing a predefined problem-solving method, and by providing a library of mechanisms with which knowledge engineers can craft new problem-solving methods, PROTÉGÉ-II permits a complete definition of all knowledge roles through the tool itself.

4. A Running Example

Now that we are familiar with the different pieces that make up the PROTÉGÉ-II system, we shall illustrate the systems's use through an example of the generation of a knowledge editor. Because PROTÉGÉ-II is not fully implemented, not all the functionality outlined in the previous sections will be demonstrated in this example.

The knowledge engineer's goal in our example will be to produce a knowledge editor for a knowledge base that will be employed by an advice system that assists physicians managing cancer therapy. The task area is that of clinical-trial management in which patients are assigned at random to one of several alternative treatment plans. The details of a treatment plan are specified by physicians through what is called a *protocol*. Knowledge acquisition in this context consists of obtaining the description of protocols, which includes knowledge about the components of the protocols, the relationships among the components, and the procedural knowledge necessary to carry out the treatment plan.

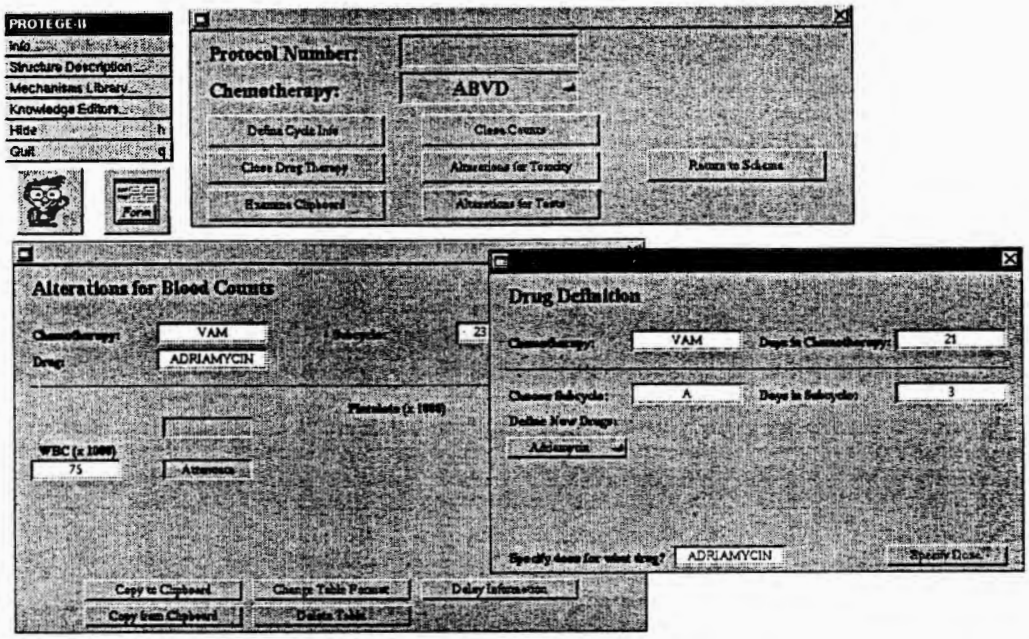


Figure 11. The form-based portion of a knowledge editor to enter protocols. This interaction style is preferred for certain types of knowledge, such as protocol data. The forms shown in this figure are part of a hierarchy of forms constructed by the knowledge engineer. The hierarchy controls the order in which the forms can be displayed, and thus guides the application expert through the knowledge-acquisition process.

Once the knowledge engineer makes a decision regarding presentation styles, a preliminary editor is generated by PROTÉGÉ-II from the task model developed and the presentation styles selected. This editor consists of a palette, or palettes, of graphical objects that can be utilized to draw diagrams in a graphical editor, and of a hierarchy of forms. The knowledge engineer then uses the forms and graphical-objects languages to impose domain-dependent constraints on the behavior of the interface components, thus adapting the editor to the requirements of the user, the task, and the domain. The PCL language, which constrains the behavior of the palette objects, predefines a number of types of graphical objects (e.g., iteration box, decision box) from which all other objects in the palette are derived. The predefined types provide a general framework to define graphical objects to capture procedural knowledge.

Figure 10 shows the palette for our example, as well as a partial protocol drawn using the objects in the palette. It also shows part of the PCL description for the palette. Notice that the declaration for the object *chemo* (for chemotherapy) has a constraint *no-connect-to* that determines that this object cannot be connected to itself. The graphical-objects compiler takes the PCL code and generates a knowledge-editor interface that automatically enforces the constraints specified in the PCL description. The compiler, however, does not prescribe a visual representation for the graphical objects that appear in the palette, although a default (not adapted) representation (where every object is a box distinguishable by only its label) is available. In Figure 10, notice how the knowledge engineer makes use of this freedom to produce highly expressive palette objects that can aid the capture of knowledge from the application expert.

The protocol of Figure10 contains several examples of additional subtyping of graphical objects made by the application expert. For example, a subtype of the object *chemo* called *VAM* was defined. Subtyping of an object in the graphical editor is allowed unless a *no-modify* constraint is declared for the object.

The application expert enters certain facts about the protocols using forms. These forms are defined by the knowledge engineer, using the FormIKA language, as part of the process of knowledge-editor generation. Figure 11 shows some of the forms associated with the protocol in Figure 10 (Protocol 2091). Using FormIKA, the knowledge engineer defines a proper hierarchy of forms that guides the application expert during knowledge acquisition. Also, since automatic updating of entry fields can be defined through the language (e.g., the name of the protocol appears in the

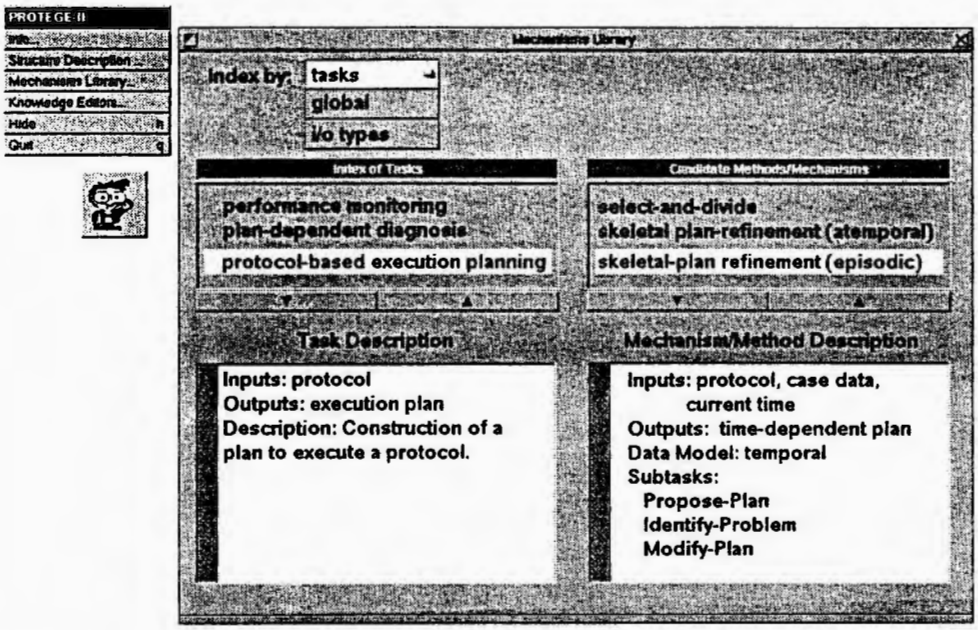


Figure 7. Interaction with the index system. The goal of the knowledge engineer is to find candidate methods or mechanisms that can be configured or assembled into a problem-solving method for the selected task.

4.1. Construction of the Problem-Solving Method

In the initial phase in the construction of the knowledge editor, the knowledge engineer uses the facilities of the library of mechanisms to look up candidate mechanisms, or methods, to solve the class of tasks at hand. Figure 7 shows the index system. The user has selected the index-by-task search strategy, and has found a suitable entry by browsing through a list of tasks. Selecting a task in this list displays both a textual description of the task, and the task's source in a separate browser. Similarly, the selection of an entry in the second browser shows textually the characteristics of that entry.

The descriptions of tasks and mechanisms appearing under the browsers aid the knowledge engineering in judging the appropriateness of the task and the applicability of the mechanism. In Figure 7, the user believes that *protocol-based execution planning* is a good match, and is investigating the applicability of the three known members of that task's source. In the domain of interest, it is necessary to manipulate temporal data (e.g., laboratory-test results, patient symptoms); thus, it is important that the mechanism selected support this type of data in its global data model. In this case, *episodic skeletal-plan refinement* is the only candidate method that fits this requirement (this information can be found in the textual descriptions). Additional details about a global data model can be extracted in the index system using the index-by-data-model search strategy.

Episodic skeletal-plan refinement is a method, as can be seen from the *task decomposition* associated with it. The user can now choose whether to configure this method in the configuration palette, or to call the assembly editor to modify the method's assembly. Figure 8 shows the configuration palette for our example. Each of the subtasks in the task decomposition is related to a multiple-selection button. The choices on the buttons are the mechanisms known to be in the subtask's source. The user can access information about these mechanisms in the index system by selecting the corresponding subtask. Substituting one mechanism for another is as simple as clicking on the proper button. However, the user also is given the option of replacing a mechanism on the palette by one not present in the buttons.

For comparison, Figure 8 also depicts the same method in the assembly editor. Notice that the assembly editor is much more flexible because, by deleting or adding mechanisms, the user can define a different task decomposition. In addition, the editor allows the user to modify the control configuration of the method dictated by the connecting links between mechanisms. The PROTÉGÉ-II user would normally move back and forth among the palette, the editor, and the index system to construct the problem-solving method. If, by using the building blocks in the library of

mechanisms, the user can custom-tailor a new method, this method will be saved and incorporated into the library for future use.

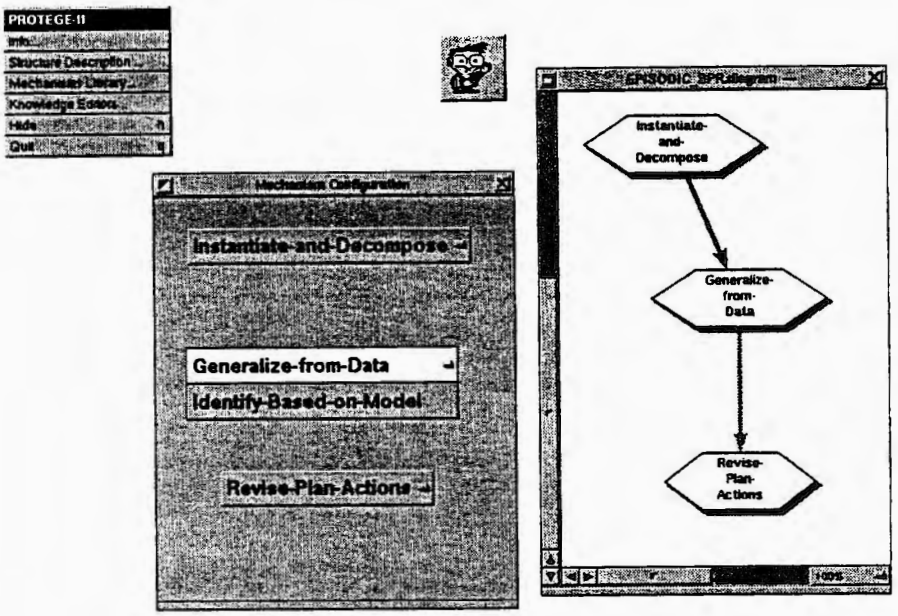


Figure 8. The configuration palette and the graphical assembly editor. The knowledge engineer can use the palette to select one of the available mechanisms for each of the subtasks in the method's task decomposition. Selecting a different mechanism using the index system, or assembling one with the assembly editor, is also possible. Notice that the assembly editor permits editing of the links between mechanisms, thus offering a level of flexibility much higher than that of the configuration palette.

4.2. Modeling of the Task

The problem-solving method of skeletal-plan refinement starts with the creation of an abstract (skeletal) solution to a problem. The abstract plan is decomposed into one or more constituent plans that are developed in more detail than is present in the original abstract plan. The constituent plans may, in turn, need to be decomposed further in a similar fashion. The process continues until a complete, specific plan for the given problem is defined. Episodic skeletal-plan refinement is a version of this problem-solving method that can operate with time-dependent data such as those in our running example.

The problem-solving method that is constructed using the library of mechanisms dictates the terms in which the task can be modeled. In our example, the task can be modeled in terms of *planning entities*, *input data*, and *actions*, as determined by the model of episodic skeletal-plan refinement [Musen and Tu, 1991]. Planning entities are time-varying problem-solution components that can be decomposed into other planning entities, thus forming a hierarchy. Input data are gathered from the environment affected by the problem. Actions are procedures that can modify instances of active planning entities. For example, in our domain of interest, the knowledge engineer can define protocols as planning entities, and can decompose such entities into other planning entities, such as chemotherapies[1] and radiation therapies. A laboratory-test result will be part of the input data, whereas drug attenuation can be an action that affects chemotherapy planning entities.

In addition, planning entities and actions can have *attributes*. The role of an attribute is to attach a value to a particular characteristic of a planning entity, or action, needed for problem solving. For example, in episodic skeletal-plan

1. Chemotherapies are groups of drugs administered to cancer patients in a particular order and in specific dosages.

refinement, all planning entities have an attribute called *duration* that determines when each entity is active in the plan. This attribute gives planing entities their temporal nature.

As explained in Section 3.3, problem-solving methods in the library of mechanisms are associated with programmatic descriptions of forms and graphical objects. The UIMS processes these descriptions to display an interface for the knowledge engineer where the task-modeling interaction can take place. In Figure 9, we can see part of the interface for episodic skeletal-plan refinement. The form shown and others in the form hierarchy permit the knowledge engineer to define the domain-specific concepts for the given task as allowed by the problem-solving method. It is in this manner that construction of a method influences the task-modeling phase.

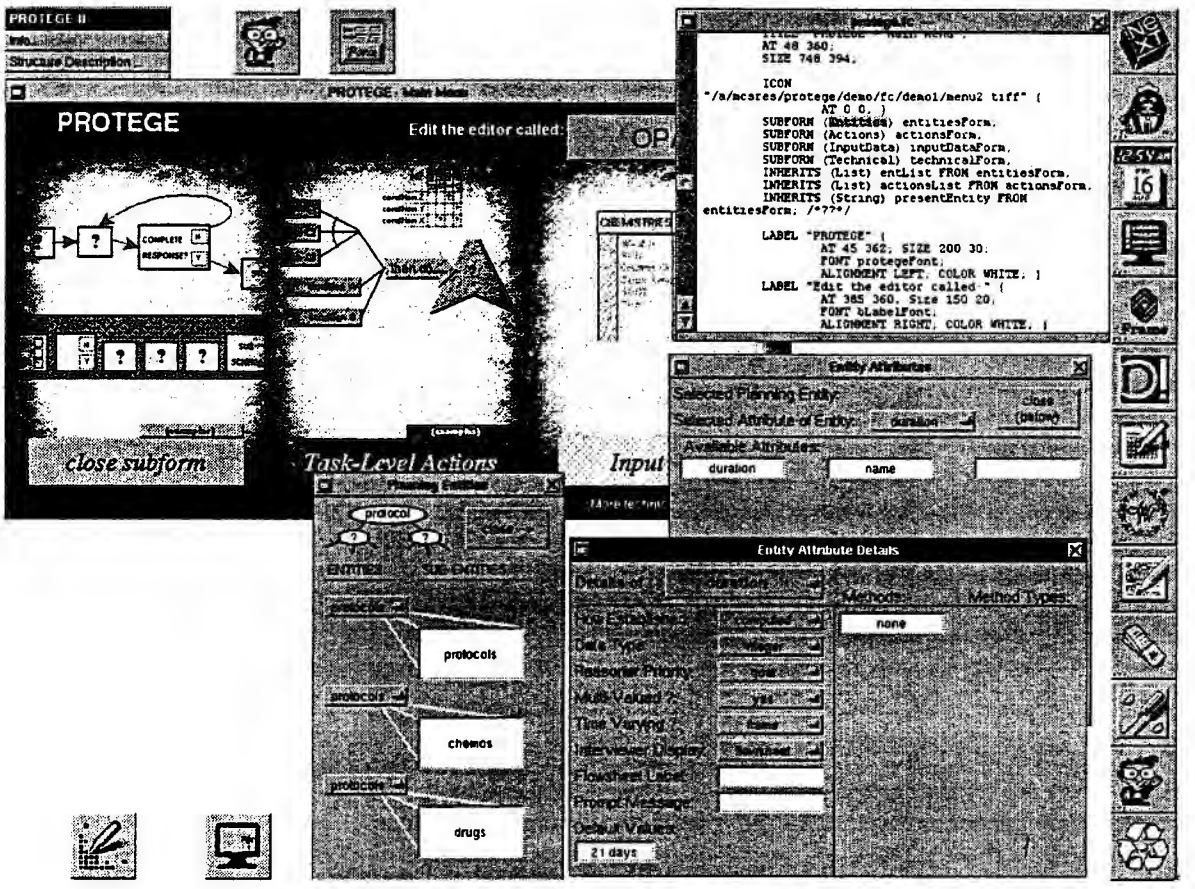


Figure 9. The interface for the task modeler. Each method is associated by design with an interface for modeling tasks in the terms prescribed by such method. The interfaces are specified using formal languages. This example shows forms to define planning entities, and attributes, for episodic skeletal-plan refinement, along with the FormIKA programmatic description of the forms.

One of the advantages of providing a UIMS is the resulting freedom in how the method requirements for task-modeling are enforced on the user. In this case, the knowledge engineer is presented with a number of forms where planning entities, attributes, actions, and input data can be defined. By associating the proper programmatic description of graphical objects with the problem-solving method, however, the user can construct the planning entity hierarchy in a graphical editor (e.g., as a tree). In fact, forms and graphical interfaces can both be associated with a given method and the user can make his own choice as to the interaction style to be employed.

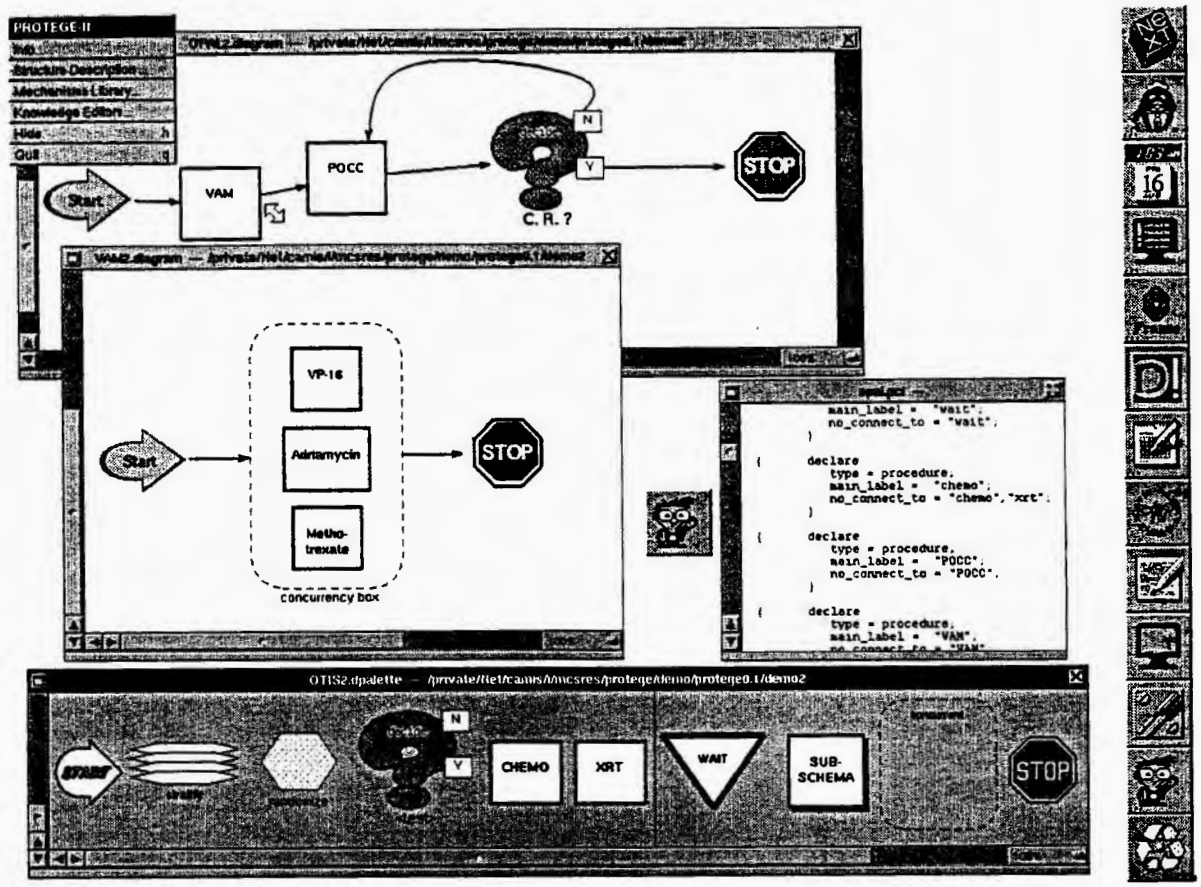


Figure 10. The graph-based portion of a knowledge editor to enter protocols. Application experts use a palette of graphical objects to construct flowcharts of their protocols. This example shows the palette, and a partial specification of a protocol, along with the associated PCL programmatic description of the palette objects. Because the knowledge engineer is allowed to stipulate the interaction style, the resulting palettes in PROTÉGÉ-II can be highly expressive, and closely related to the domain of interest.

4.3. Generation of the Knowledge Editor

The phase of knowledge-editor generation requires a specification of the interface between the application expert and the knowledge-acquisition system. Therefore, the knowledge engineer must define the contents of the interface, the presentation style of these contents, and the behavior of the generated interface components. Normally, knowledge-editor generation will comprise the following steps:

1. Determine the types of knowledge to be acquired
2. Decide the most appropriate style of presentation for each type of knowledge (form-based or graph-based)
3. Generate automatically a preliminary knowledge editor.
4. Write programmatic descriptions of the forms and graphical objects to be used, which constrains the behavior of the interface components
5. Define the connections between graphical objects and forms

In our continuing example, the knowledge about a protocol that will be acquired can be procedural or factual. The protocol is, by definition, a procedure. Clearly, however, many details about the protocol—such as types of drugs, and dosages of drugs—are factual. Since physicians typically draw flowcharts to provide high-level descriptions of protocols [Musen et al., 1987], it is appropriate to acquire this procedural knowledge graphically while obtaining facts about the protocol through the use of forms.

subforms automatically), the knowledge engineer can simplify the expert's work by designing forms such that there is no need to repeat entries.

The link between the protocol graph and the protocol form is maintained by the structure manager. This subsystem accesses a database that contains information on all interface structures and their directional links. Interface structures are added to the database from the FormIKA and PCL declaration of the structures. Links are added by the knowledge engineer. The structure manager has access as well to information about the constructed problem-solving method and its associated forms and graphical objects. The method itself is viewed by the structure manager as a graphical object that has inputs and outputs. Thus, to establish that a protocol is an input to episodic skeletal-plan refinement, the knowledge engineer requires no more than establishment of a link between that planning entity and the appropriate method input.

5. Discussion

For the past several years, researchers have been studying the use of models of problem-solving methods to construct knowledge-acquisition tools [McDermott, 1988; Musen, 1989a]. The resulting applications, although successful to some extent, have revealed serious limitations with this approach. The domain-independent nature of the models makes it difficult to express the role of knowledge that depends on domain-specific considerations. In addition, the resulting knowledge-acquisition tools are restricted to the class of tasks that the model can represent.

Whereas scientists have quickly recognized this problem, considerably less attention has been given to the human–computer interaction issues associated with the specification of interfaces for knowledge-acquisition tools built from models of problem-solving methods. A major difficulty in this area is the inability of interfaces derived from domain-independent models to express and capture knowledge that is task- and domain-specific, or to show any adaptability to the needs and requirements of the domain experts who use these tools. These limitations are crucial because regardless of the appropriateness of a given model, a tool derived from such model is not useful if its interface is not suitable for communication with the user.

There is a clear paradox in the desire to build knowledge-acquisition tools that are based on domain-independent models but that simultaneously present interfaces that are task-, domain-, and user-dependent. Our approach to this challenge is two-pronged. On one hand, we are investigating the use of building blocks, called mechanisms, that can be used to construct problem-solving methods. By using mechanisms, we hope to eliminate the limitation of single-method architectures and to be able to define both domain-independent and domain-dependent knowledge roles. On the other hand, we are developing a user-interface management system that would allow us to adapt interfaces derived from the constructed problem-solving methods to the requirements of individual tasks, domains, and users. This capability is crucial in PROTÉGÉ-II, which is a metatool where interfaces for the resulting knowledge-acquisition tools are generated in great part automatically.

On the first front, our current endeavor is to identify mechanisms from expert systems previously developed in our laboratory [Tu et al., 1989; Musen, 1989a]. We are building the library of mechanisms as an effective architecture to index the identified mechanisms based on their components (see Section 3.2). The design of the library of mechanisms is similar to that of a library for books. The knowledge engineer can use multiple search strategies to find candidate mechanisms that can be further configured, or assembled, into a problem-solving method. Through the use of PROTÉGÉ-II, we expect to gain a better understanding of the indexing strategies that are most useful for knowledge engineers. Separately, we are studying the control problems that arise in the assembly and configuration of mechanisms and the graphical means to provide for the construction of problem-solving methods. The graphical assembly editor, which is part of the library of mechanisms, will allow us to experiment with the data flow and control flow among mechanisms. We will investigate the extent to which the mechanisms in the library can be combined effectively in such an editor to form new problem-solving methods.

On the second front, we are examining means to generate adaptable interfaces in PROTÉGÉ-II, and are studying the effect of the adaptability requirement on the definition of mechanisms. One of our first results was the identification of an interface specification as a necessary component of a mechanism. In PROTÉGÉ-II, each mechanism defines what interface specifications can be used to model a task under the terms of such mechanism (see Section 4.2). In addition, the knowledge editors generated by PROTÉGÉ-II from the task model are considered preliminary. The user-interface management system provides the knowledge engineer with the means to adapt these preliminary editors to individual tasks, domains, and users. In this manner, we bridge the gap between an editor derived from a domain-independent

model and an editor tailored to the requirements of a single task, domain, and user. Although the current version of PROTÉGÉ-II supports only form- and graph-based interaction modes, we are merging FormIKA and PCL—our formal languages for interface specification—into a single, more general, language that will support additional modalities and drawing techniques (e.g., jigsaw puzzle). We hope not only to gain additional flexibility in the generation of interfaces, but also to study approaches for mechanism assembly different from the nodes-and-links paradigm.

PROTÉGÉ-II's view of mechanisms as reusable programming constructs parallels, in certain aspects, that of Spark [Klinker et al., 1991]. Perhaps a clear contrast between these two systems lies not in the particular techniques used in the manipulation of mechanisms, but rather in the different approaches to the interaction between the user and the system. Whereas PROTÉGÉ-II aides a knowledge engineer in the construction of knowledge-acquisition tools, Spark guides a nonprogrammer through the process of selecting an assemby of mechanisms. PROTÉGÉ-II has the relative advantage of presenting information to a knowledgeable user who can make informed decisions. Spark, in contrast, relies heavily on the system to make critical choices.

One system that recognizes the central importance of domain-specific interfaces for knowledge-acquisition tools is DOTS [Eriksson, 1990]. In DOTS, knowledge engineers can construct interfaces for knowledge-acquisition tools by specifying a window-system layer on top of the Interlisp-D system. The process, however, is mostly manual. The intent of the user-interface management system in PROTÉGÉ-II is to relieve the knowledge engineer from having to specify in its entirety the interface for each knowledge-acquisition tool developed. The preliminary knowledge editor produced from the task model is fully usable and the adaptation of this preliminary editor is greatly facilitated by the existence of formal languages that allow the stipulation of constraints in a declarative fashion.

Our emphasis on the composition and features of the interfaces presented by PROTÉGÉ-II derives from our previous experience with PROTÉGÉ. In knowledge-acquisition tools, the nature of the interface—where knowledge is presented to and elicited from users—may be as important as the means used to model the class of tasks at hand. The adaptability of this communication channel to individual tasks, domains, and users has not been given sufficient attention in the research community and is one of the key aspects of PROTÉGÉ-II that we are studying. The challenge remains, for us as well as for other groups, to develop knowledge-level architectures that can generate knowledge-acquisition tools that do not overlook the vital link to the source of knowledge: the interface to the application expert.

Acknowledgments

This work has been supported in part by grant LM05157 from the National Library of Medicine, by grant HS06330 from the Agency for Health Care Policy and Research, by a grant from the Institute for Biological and Clinical Investigation at Stanford University, and by a gift from Digital Equipment Corporation. Computer support was provided in part by grant RR05353 from the Biomedical Research Support Grant Program of the National Institutes of Health, and by the SUMEX-AIM resource, supported by grant LM05208 from the National Library of Medicine.

We thank John Dawes, Josef Schreiner, Yuval Shahar, and James Winkles for their insightful comments, Andrew Bennett for his work in developing FormIKA, and Lyn Dupré for her extensive editing of a previous draft of this paper.

References

Bennett, J. S. 1985. ROGET: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning* **1**:49–74.

Bennett, A. 1990. *A form-based user interface management system for knowledge acquisition.* Master's Thesis. KSL-Report 90-43, Knowledge Systems Laboratory, Stanford University, Stanford, CA.

Chandrasekaran, B. 1986. Generic tasks for knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* **1**:23–30.

Clancey, W. J. 1985. Heuristic classification. *Artificial Intelligence* **27**:289–350.

Eriksson, H. 1990. Meta-tool support for customized domain-oriented knowledge acquisition. In *Proceedings of the Fifth Banff Knowledge-Acquisition for Knowledge-Based Systems Workshop*, Boose, J. H. and Gaines, B. R., editors., pp. 6.1–6.20. Banff, Alberta, Canada.

Friedland, P. E., and Iwasaki, Y. 1985. The concept and implementation of skeletal plans. *Journal of Automated Reasoning* 1:161–208.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., and McDermott, J. 1991.Usable and reusable programming constructs. *Knowledge Acquisition* 3:117-135.

Marcus, S. and McDermott, J. 1989. SALT: A knowledge acquisition tool for propose-and-revise systems. *Artificial Intelligence* 39:1–37.

McDermott, J. 1988. Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems*, ed. by Marcus S., pp. 225–256. Boston: Kluwer Academic.

Musen, M. A., Fagan, L. M., Combs, D. M., and Shortliffe, E. H. 1987. Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man–Machine Studies* 26:105–121.

Musen, M.A. 1989a. *Automated Generation of Model-Based Knowledge-Acquisition Tools.* London: Pitman.

Musen, M. A. 1989b. An editor for the conceptual models of interactive knowledge-acquisition tools. *International Journal of Man-Machine Studies* 31:673–698.

Musen, M.A., and Tu, S.W. 1991. *A model of skeletal-plan refinement to generate task-specific knowledge acquisition tools.* KSL-Report 91–05, Knowledge System Laboratory, Stanford University, Stanford, CA.

Newell, A. 1982. The knowledge level. *Artificial Intelligence* 18:87–127.

Puerta, A. 1990. L-CID: *a blackboard framework to experiment with self-adaptation in intelligent interfaces.* Ph.D. Dissertation. Technical Report USCMI 90-esl-6, Center for Machine Intelligence, University of South Carolina, Columbia, SC.

Rissland, E. 1984. Ingredients of intelligent interfaces. *International Journal of Man–Machine Studies* 21:377–388.

Steels, L. 1990. Components of expertise. *AI Magazine* 11(2):30–49.

Tu, S., Shahar, Y., Dawes, J.,Winkles, J., Puerta, A., and Musen, M. 1991. A problem-solving model for episodic skeletal-plan refinement. In *Proceedings of the Sixth Knowledge-Acquisition for Knowledge-Based Systems Workshop*, ed. by Boose, J. H. and Gaines, B. R., Alberta, Canada, in press.

Vanwelkenhuysen, J., and Rademakers, P. 1990. Mapping a knowledge level analysis onto a computational framework. *Proceedings of the Tenth European Conference on Artificial Intelligence*, pp. 661–664. Stockholm, Sweden.

A Theory of Local and Interstitial Knowledge

Alain T. Rappaport[1]

Neuron Data
156 University Avenue
Palo Alto, CA 94301, USA

atr@ml.ri.cmu.edu

This paper presents some elements and the rationale underlying a theory of local knowledge. Local knowledge encompasses the local processing of knowledge in the context of human and computer problem solving activities. In contrast with monolithic knowledge-base design approaches which emphasize task-oriented processes, it merges well with modern software engineering paradigms. Interstitial knowledge is an extension of local knowledge. We enumerate a set of cognitive principles at the basis of the theory and explore some related computational aspects.We then elaborate on the processing and acquisition of local knowledge and its execution in relation with other software. While knowledge processing is at the center of the theory, the latter has implication which we believe extend beyond the traditional AI issues into the general software domain.

DEFINITION

Local knowledge consists in the opportunistic acquisition and intervention of chunks of knowledge with the effect of elevating the level of abstraction of one or several processes closer to the problem being ultimately solved. Local means "belonging to" or "affecting a particular place" (Ehrlich et al., 1980). The present theory emphasizes the pervasiveness of knowledge processing, either specific or general, in every conceivable place, even for very simple purposes. Opportunistic is used here in a broad sense, to indicate the timeliness of local knowledge with respect to the current activity. There is a wide variety of actions in which knowledge is involved which may not fit classic categorization of tasks and may not even be clearly definable. Implementing local knowledge processing on a significant scale results in a global 'background' processing which brings existing processes into the context of a higher level of abstraction. Knowledge acquisition becomes an effort closely related to local requirements and representations; it must be integrated with the other data processing techniques relevant to the local purpose. The theory addresses the nature of problems, the knowledge structures involved and the execution processes. Interstitial means from "a small intervening space" (Ehrlich et al., 1980). This term illustrates the computational implications of pervasive local knowledge in a more general computing environment.

---

[1] also Adjunct Research Scientist, Carnegie Mellon University, School of Computer Science, Robotics Institute Pittsburgh, PA 15213.

PRINCIPLES

We identify in the following paragraphs seven important principles underlying the theory of local knowledge.

The Observer/Problem Relation

The definition of a problem is not independent of its observer (Rappaport, 1988). In puzzle-solving, the observer can be removed by expressing a near observer-independent strategy. We postulate here that it is very rarely the case in problems encountered in normal human activities. The use of knowledge is highly dependent on the context, part of which is defined by the observer's intentions and actions. The fact that a given state represents a solution to the problem is often a function of the observer's mental model and of the constraints within which the reasoning takes place. This approach is directly compatible with the notion of good enough solutions (Simon, 1981). Furthermore, it emphasizes observer-related systems rather than process-directed architectures. Including the observer into the problem solving process means changing the level of abstraction of the latter.

Open Problem Space

To the traditional problem space definition (Simon and Lea, 1974) we add the fact that, in human activities again, the actual space and therefore set of states is in most cases open, with ill-defined boundaries, rather than closed and well defined. The set of states and of operators the problem can be in is not easily definable and changes with time. The solution is not a computational or mathematical solution but rather a satisfactory state which allows for actions to be taken. The definition of satisfaction is a state defined by both internal and external parameters.

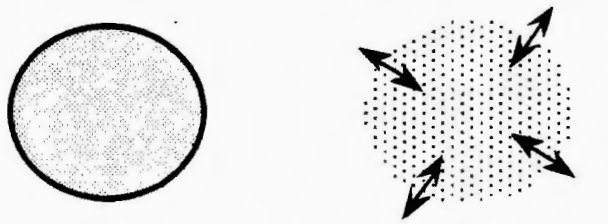


Figure 1. Bounded rationality vs. open rationality

Figure 1 graphically illustrates the open problem space. To reach a satisfactory solution one has recourse to unexpected data and/or operators. From the initial assumption of a bounded rationality we can derive states and operators for a given problem space. However, a natural consequence of exploring that very space in the open problem space framework is to modify the initial boundaries of rationality. New states and operators, new knowledge must be described. Problem-solving is thus as much an exchange or communication process as a modification of internal states of operators. We talk about open rationality rather than bounded to express the changes in the boundaries.

Levels of Scale

Once the problem is viewed as open, the set of constraints could be considered infinite. In problems with large problem spaces learning mechanisms such as chunking allows for a renormalization of the problem, i.e. a reduction in scale and complexity to a manageable set of

parameters while preserving the fundamental properties, performed by the observer (Figure 2).

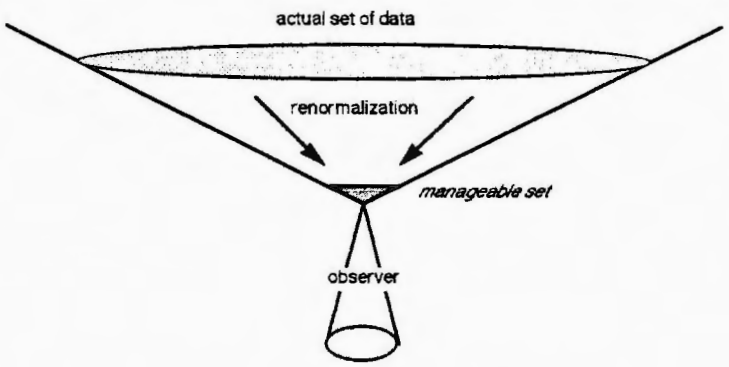


Figure 2. The observer's focus of attention.

Such reductions in scale are critical and apply at all times in open problem spaces. Should an observer reason upon some data, the latter needs to be reduced to an expression comprehensible by that observer. For instance, induction from large databases are such a reduction process when it can generate the smallest and most representative set of knowledge elements (Gaines 1989, Catlett 1991).

In figure 2 the observer can be either a human or a computing agent. We view any process which accesses knowledge processing as an agent with its own focus of attention and its own level of abstraction. The reduction in scale is necessary to deal with the limited resources at the attention level of human agents, but computing agents also have such a focus.

Continuum vs. Categorization

Problem solving most often comprises a mix of perceptual, analytic and synthetic processing modes. By categorizing too narrowly and perhaps arbitrarily the problems solving tasks, we reduce the power of systems unless the problem can exactly fit the model. There is no *task* theory with universal value by virtue of the context dependence and dynamics of human problem solving. To reflect this, we use 'activities' rather than tasks. This context dependence requires that the approach to problem solving is flexible, using different mechanisms which can be recombined to provide for more adapted and powerful tools to proceed with.

As a convention, we graphically represent a theory or a problem solving mechanism by a *cube*. A larger cube is a higher level theory, such as task-dependent theories. At the other end of the spectrum are mechanisms such as cognitive primitives (Rappaport, 1988) and lower level routines. Figure 3 illustrates the possible equivalence between a single theory and a combination of lower granularity mechanisms. The association of such mechanisms can provide with a similarly adequate theory but it is more powerful in that recombinations can be made to adapt to the changing nature of the problem. Even if each mechanism is simpler than the large theory, their association is likely to perform better overall since it can match the problem or situation rather than require the latter to match the theory (an often unlikely event).

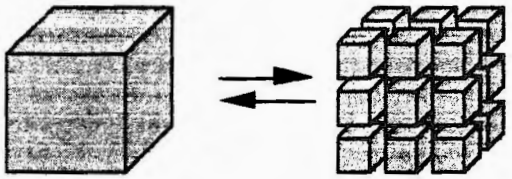


Figure 3. An asssembly of modules is equivalent but more versatile and adaptive than a single well-bounded theory.

The single cube is a monolithic approach, contrasting with the polylithic approaches to problem solving. Task level theories have little elasticity and require that the problem fit their framework. Monolithic theories have a non-flexible scope, i.e. the shape of the problem cannot vary much or must not (figure 4). Furthermore, the lack of flexibility requires that the problem is somewhat closed.

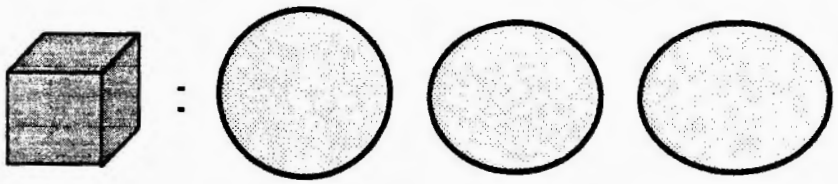


Figure 4. Only slight modifications of the problem space are allowed by monolithic architectures.

To deal with open problems which by definition can change rapidly and widely, a lower granularity level is required to recombine theories so they adapt to the nature of the problem (fig 5).

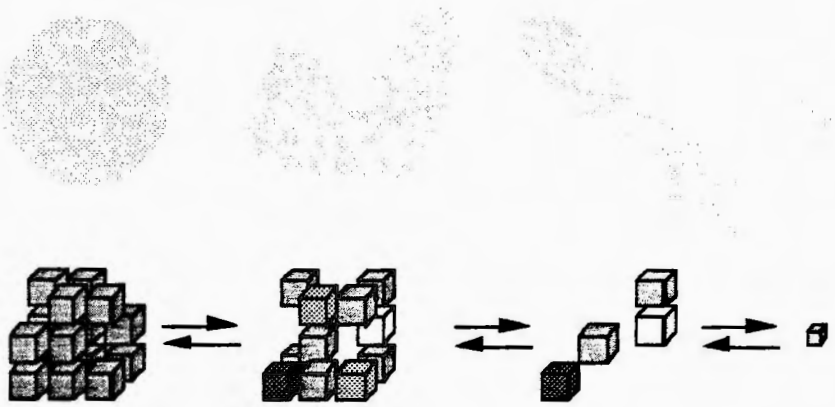


Figure 5. Changes in modules to adapt to the changing problem.

As the shape of the problem changes, different sets of mechanisms are brought into play. Furthermore, the local arrangement of one or several theories allows the problem solving activity to occur in conjunction with other processes. Local knowledge is local in purpose and time, but it is reuseable and can intervene in different places at the same time, for the same or different purposes.

Integration

Just as human actions are in context, knowledge processing must be in context, providing actions which will modify the relevant environment. In computing terms, this equates to integration with other processes. Integration in the biological sense is expressed by the tight coupling of the higher level function with both the perceptual and effector components of higher organisms. This architecture must be mapped into computer systems. Knowledge processing is only one aspect of the overall activity but as much as it may represent a small amount of the total processing, it must be very tightly integrated. Figure 6 illustrates that the openness of the problem space (left) is represented in the integration of the knowledge systems with other modules.

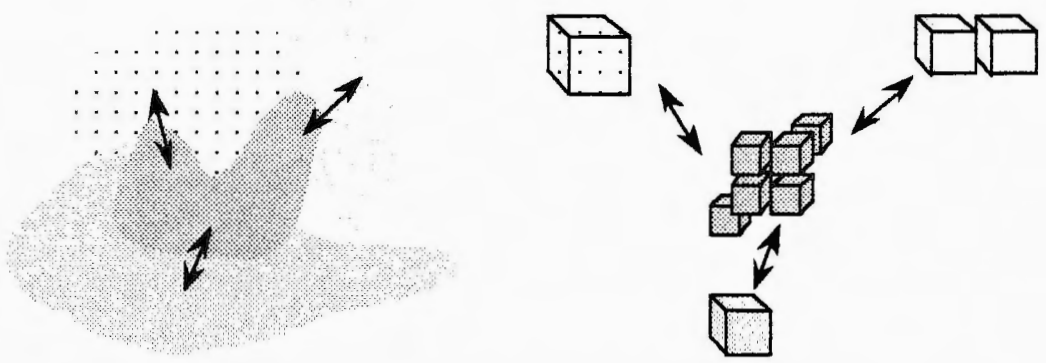


Figure 6. To the interacting domains corresponds interacting libraries.

Thus, the open model of recombining problem solving mechanisms extends to the rest of the software, traditional or not, which may be involved in getting to a satisfactory solution.

Abstract Representations

Knowledge starts where raw data ends in the sense that any interpretation of data already constitutes a knowledge processing activity. The role of knowledge is to put the data or facts into use at a higher level of abstraction. This is a transformation process used to make sense of the information with respect to a particular context. .

The knowledge processing in figure 6 needs to be *relevant* to *many* different subsystems. We are familiar with abstract data types, but this type of integration requires the existence of *abstract data structures*, i.e. data structures which can be universally mapped into others, more specific. Abstract data structures must not be overspecified and yet cannot be too general so as not to carry a large overhead.

Figure 7 illustrates the communication between processing units and the knowledge processing one by means of an abstract common data structure (ACDS). DS a, DS b and DS c stand for the data structures of units a, b and c. The interactions are *mapping* interactions where properties are mapped from the specified data stucture into the flexible common representation.

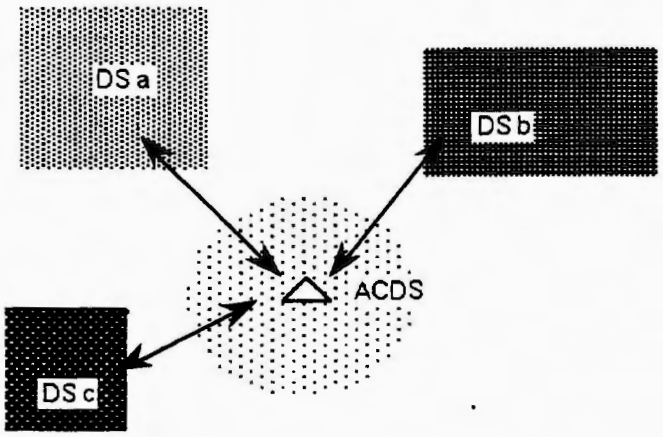


Figure 7. An abstract common data structure is shared by different programs.

The existence of such ACDS *allows for* the integration process and plays a critical software engineering role. Not only does the ACDS allow for reasoning *about* the original data structures but by virtue of its being *common*, it allows for communication between the otherwise non communicating software a, b and c. The mapping process requires that the knowledge processing module can create the ACDS.

Collective Symbolic Computing

The impact of knowledge processing is greater when considering the collective effect of many local knowledge structures influencing as many domains as possible. Knowledge is often subjective and local. The collection of knowledge affects higher levels of organization and is responsible for the knowledge-based efficiency. As seen above the integration of local knowledge to existing environment creates a vast array of potential added values on top of the original process. This integration can only take place if the software engineering solution is elegant, simple and at an acceptable level of granularity to allow for maximum flexibility. Emerging properties was so far a principle reserved to collections of very small related units performing a search function. We are extending this to the social and economic properties emerging from the use of local knowledge processing pervasively in relation with virtually any computer process.

CONTROL STRUCTURES FOR LOCAL KNOWLEDGE

Local knowledge is local in time and space. It is the opportunistic instantiation of knowledge level structures in a more global context to "solve a problem", which means in general moving ahead with one's activity. An activity is a broader concept than a task. The definition of a task has often implied pre-defining a problem space, creating an algorithm to perform in that space and then fitting a problem to the latter. In fact we instantiate the right mode of thinking at the right time, making trade-offs, commitments, revisions and other dynamic decisions. For instance, a mechanic may be looking for the cause of a breakdown which seems like it is a diagnosis tasks. In fact he is trying to satisfy a customer, optimizing his time, managing multiple car repairs, training an apprentice etc... Likewise, in real life crisis situation on an aircraft carrier the usually normal procedures are diluted in the sudden context of an emergency. The recent deployment of a large AI-based architecture has shown that, in the context of the real-world action, humans tend to take

ad-hoc and indeed good enough rather than exact solutions faster than this reasoning powerful architecture could provide an answer (Sloane, 1991). The theory of local knowledge emphasizes the need to act quickly, to decentralize systems, to provide flexible architectures acting at different levels of abstraction.

Control Levels

The first level of control called atomic level is also the more general. One single rule or one chunk of knowledge is always worth more than none. Hence, the 'injection' of a single chunk of knowledge in a given context should be exploitable immediately or with a certain priority with respect to current events. At this atomic level, the ability to make even one inference may be quite important and bring significant added value. Clearly, to make one inference one does not need a deductive engine, a sophisticated pattern-matcher or other. At this level, the knowledge system may be fully opportunistic. The system is either in the background, polling for information, or the process in question knows which of its own events can benefit from the heuristics. One can easily imagine 'zillions' of such atomic chunks of knowledge triggered in appropriate situations.

Interactive control is an extension of the atomic level where information is required from either the user (human) or from other processes for the chunks of knowledge to be fully matched and executed. Interactivity is dependent upon the nature of the local activity of the user.

Agenda-based control structures are the next extensions to an ubiquitous atomic level. As simple as the atomic is, reaching multiple conclusions (actions) will generate conflicts to resolve. The natural type of architecture is the agenda-based control structure made of adequate primitives (Rappaport 1988) where events are inserted both in the stack of the knowledge system and in the event loop of the associated processes. The management of the relation between the event loops is critical to the integration.

An approach based on building blocks or sets of operators to map the actual problem solving activity into a representation has been very successful (Rappaport, 1988) - example are mentioned later in this paper. It consists in fact in providing a general framework to build ad-hoc knowledge-based systems for a wide variety of domains. Other examples have been developed along similar lines, emphasizing simplicity and efficiency (Compton and Preston, 1990). Much knowledge-based activity consists in correctly indexing a body of data or problem states. It is important to recall the cognitive relevance and usefulness of EPAM-like architectures (Simon and Feigenbaum, 1984). The implementation of such discrimination nets is a subset of what is allowed by the cognitive primitives approach. Cognitive primitives can be extended to larger building blocks performing more specific algorithms.

The next level is when a given control structure seems to resolve a macro-operation compared to the atomic level or association of chunks of knowledge. These macro-operation are situation specific control structures. They can be either acquired automatically as in chunking or macro-operator acquisition or can be built in an ad-hoc fashion. For example, neural net routines, deductive mechanisms, analogy-based reasoning or others would be called into when appropriate. Thus in order to solve a problem the system may resort to different control structures at various times, for different purposes. Such systems may be very narrow in scope but allow to focus precisely on a very specific body of knowledge for a specific part of the activity.

The following figure is an abstract illustration of the gradient of control structures described above.

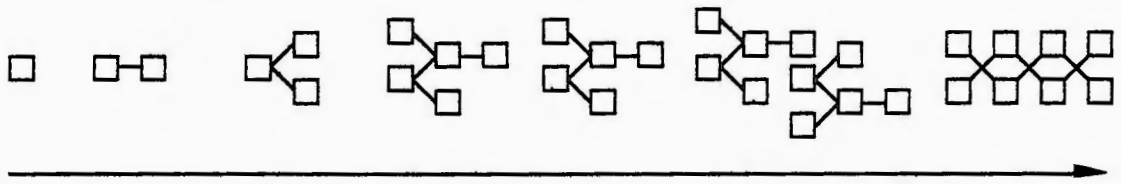


Figure 8. From atomic to specific knowledge.

On the left of the diagram with an atomic chunk of knowledge (square) which in itself may be instantiated opportunistically and requires a minimal control structure. For example, `(if document references graphics then link graphics module)` can be an opportunistic rule invoked while using a document editor. As the gradient progresses to the right, control structures become more domain or task specific. For example, the diagram at the far right could be a scheduling module.

At the far left, a thread processing the knowledge level could simply interact with another thread, looking for a match with the current situation. However, as architectures become more specific, they become 'packaged' and interact via other mechanisms.

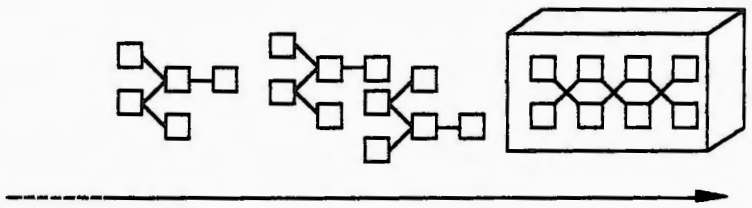


Figure 9. Specific knowledge can be packaged into libraries.

In the open problem space perspective, the external world is represented by other modules. They are part of the problem solving activity as they perform some actions which will lead to the expected level of satisfaction (again, problem solving is reaching a "satisfactory" state, or simply put: moving ahead). The event-driven architecture must also encompass subsumption relations, message-passing and other necessary characteristics of the adequate representation.

Local atomic or agenda-based knowledge can control the execution of other modules involved in the activity (the interstitial role). They can also be involved in a very local aspect of the computations within one such module.

Hence, we emphasize the notion of embedded levels of scale where the same computational structures can intervene, as illustrated in figure 10.

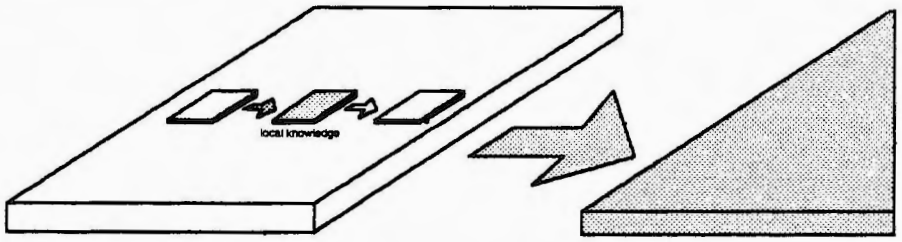


Figure 10. The same structures are used to affect different levels of scale.

SOFTWARE ARCHITECTURE ISSUES

Software engineering is a critical issue since the validation of the theory is based on the observation of actual implementation. What makes a real problem difficult, in light of the open problem space principle is not only the amount of knowledge involved and the search space size (Kotovsky and Simon, 1989) but also the integration factors. The general building block approach and the ACDS are highly compatible with more recent methods of programming and the integration of knowledge-based systems development into the mainstream software development methodologies (Meyer, 1988; Boehm et al., 1988). In this section we describe some of the theory and its relation to the cognitive aspects described earlier.

Abstract Common Data Structures (ACDS)

To describe the genesis of the ACDS we will adopt a programming perspective.

A complete procedural program P can be described as a set of functions organized in an inclusion tree. The higher in the tree, the more abstract the function/procedure can be considered.

In figure let the node A represent a function of program P. Let node B represent another function. The function C=A∪B is an abstraction of functions A and B.

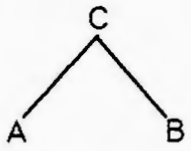


Figure 11. C is an abstraction of A and B

In another notation, if `a ()` and `b ()` are two functions of program P and ';' denotes the sequence operation, then the construction `a () ; b ()` or the following encapsulation:

```
c ()
{
      a ();b ();
}
```

denotes a higher level functional or procedural abstraction and would appear as such at a higher

level in the program tree, as in figure where nodes A and B stand respectively for functions `a()` and `b ()`.

As an example, `a ()` = draw a circle, `b ()` = fill object using current pattern, `c ()` = draw a filled circle using current pattern.

Programs are obviously much larger than the one figuratively shown above and are often created by independent authors. Therefore, it is an illusion that any two program should have a large enough intersection of its abstractions to communicate and integrate effectively. Yet, they all possess various levels of abstractions, all have data structures and data types, functions and share many other structural aspects but not functional (except for on/off and other very general functions). If P1 and P2 are two programs and thereby sets of commands, then it is likely that $P1 \cap P2 \approx \emptyset$. Much like two cells in the body serve very different functions and yet they are essentially made of the same basic chemical elements, use the same type of energy and have have more mechanisms in common than not. What makes their specificity though must be designed within this larger common framework. Specificity has to be designed in a common infrastructure and in local places to generate an organism. The same of the generation of an effective application.

If programs must be implemented independently and yet must have means to communicate, then the communication must be some other program, external but which universally maps, to a given extent dependent of the function, to other programs.

The communication we are concerned with is at the representation level. We derive from the above considerations that an abstract common data structure must be *external* to the programs which require more specific mechanisms.

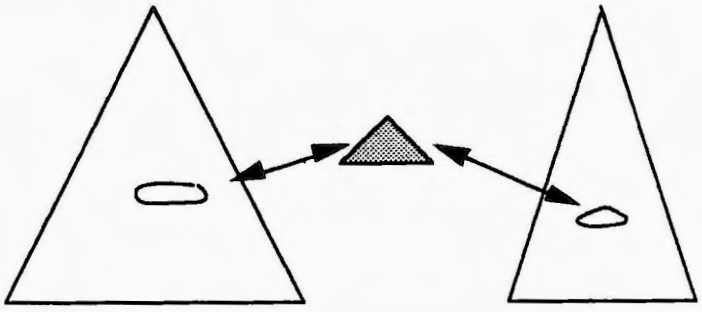


Figure 12. Two programs linked by an ACDS each at a different level of abstraction. The new data structure is 'interstitial' in that it exists outside the two programs

By definition, the manipulation of ACDS is at a higher level of abstraction than most data structures. it must be noted that there are different *types* of ACDS but we assume one type for the sake of simplicity. They are programmed at the knowledge level.

Local/Interstitial Knowledge

The local knowledge creates a 'milieu' in which other software are linked together and constitute a local environment. Local knowledge re-establishes the semantics of the programs which is

significantly lost in the design process (the program becomes a set of function).

Local knowledge at the atomic and agenda level acts as backbone guiding the correct execution of and communication between libraries. By analogy with the interstitial milieu between the cells, we speak of an interstitial knowledge layer which holds together the locally designed software architecture intended to serve the user's purpose. Local knowledge represents a new layer of software, similar to an operating system at the intentional, user level (useware). What floats in this space can be either very general, common sense knowledge or very specific atomic or agenda-based knowledge bases invoked either in a fully opportunistic mode (by an 'interstitial' engine or program) or driven by other events (from the set of libraries involved).

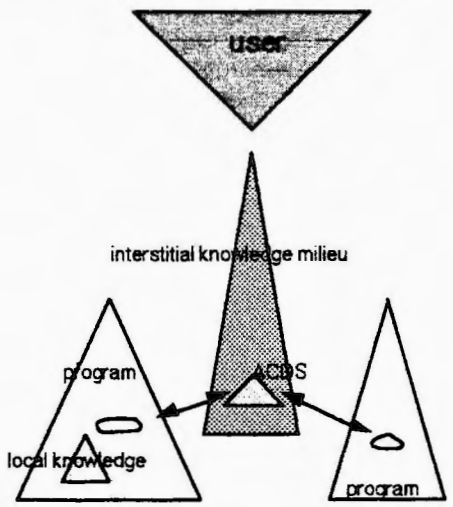


Figure 13. The user's level of abstraction can be reached by embedding both local and interstitial knowledge in the otherwise knowledge free architecture.

Figure 14 shows the evolution of software to come taking into account the ACDS and the library concepts. The present stage A can be called monolithic. These software do not naturally communicate and it is difficult to make use of their architecture for any knowledge processing or acquisition. By creating local knowledge processing capabilities that can be integrated and also allow for communication via abstract common data structures, a new stage B. The next stage C, polylithic is when software is dynamically created by linking libraries according to the user's needs (human or computer) to serve the local purpose. The knowledge processing is relevant to the manipulation of the ACDS as well as the local execution of the libraries involved.

Casual Integration

The library architecture with an application programming interface allows for the definition of clear entry points into the knowledge processing system and the linking to other object files (dynamic or not). The modularity has many key advantages. All software is going to be designed as libraries, but this concept goes beyond the packaging issues. In short, the precise boundaries of a software as we know them should disappear. It is an interaction with the user which determines which building blocks (libraries) must be assembled so as to satisfy his/her needs. As for local knowledge, libraries can represent very different levels of scale.

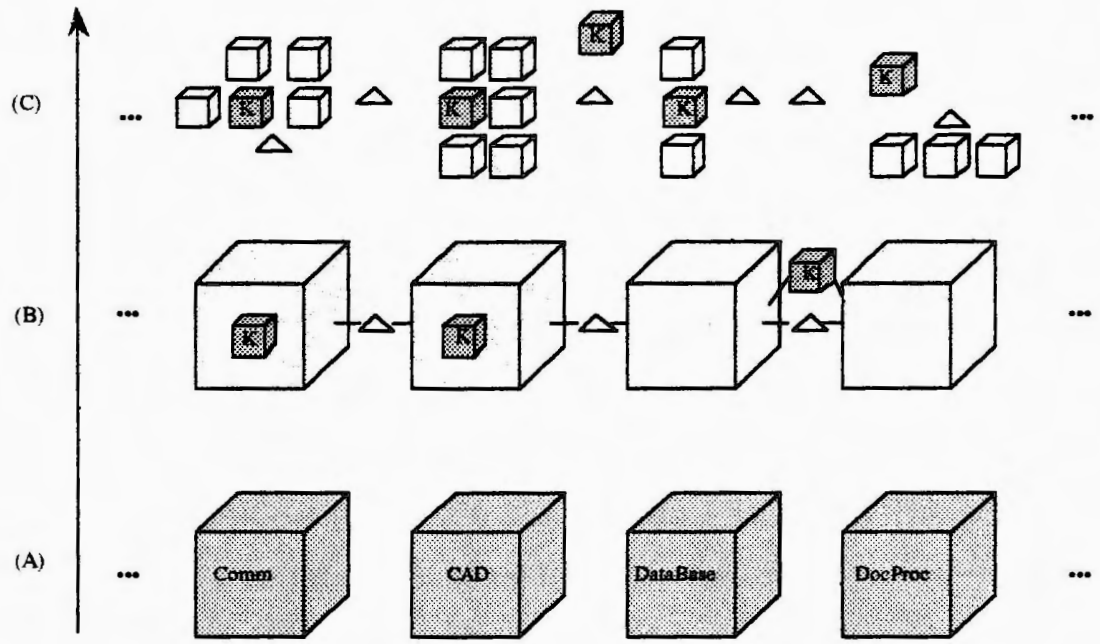


Figure 14. Evolution of software architectures.

It should be possible to integrate almost any software with another, to take any new off-the-shelf software and add the knowledge processing and acquisition capabilities to it. Several examples have already been developed using the library architecture. NEXTRA was integrated both with the NEXPERT environment and hypermedia tools (Rappaport and Gaines, 1990) to provide for an integrated environment for knowledge acquisition by either automated or manual induction, or consensus analysis. More common software can be turned into knowledge acquisition front ends to tools such as NEXPERT (Grashion, 1990). CATIA (solid-modelling) was enhanced with the NXP Library using objects as a common data structure between the knowledge engine and the custom database. ARC/Info, a geographic information system was integrated with the NXP library to provide intrepretative and analytic power on top of geographic data visualization for planning purposes (Djokic and Maidment, 1991).

The software engineering aspects of local knowledge and the relevant theories spans from the classic object-orientation to a super-object orientation which relates the software architecture to the intentional, activity levels of the user.

ACQUISITION OF LOCAL KNOWLEDGE

To approach the knowledge acquisition dimension we would like to briefly describe three examples of knowledge systems implementations which relate to the local and interstitial knowledge perspectives. They are representative of a new generation of applications being developed.

1. Djokic and Maidment (1991) have integrated the expert system library with a geographic information system (GIS). While the GIS provides a natural visual metaphor to express the modelling of urban stormwater, the knowledge processor allows for a concurrent analysis of the

terrain, looking for "missing links". Prior to this knowledge-based function, "a procedure is initiated that creates 'structures' to represent overland flow". These structures are in fact objects, or ACDS, with far less properties (only the necessary ones) than the original ones. One can imagine that the city planners can have interactive tools relating such models to policy issues in an integrated, organization-wide knowledge infrastructure. Several other such examples based on a GIS/NXP link have now appeared in the literature.

2. Averback (1991) has designed the requirements for a "Knowledge Workstation" in the context of building tools to analyze vast amounts of data for the human genome project (HUGO). The objects represent chromosomes. They can be stored in a relational database or come from some other tool (scanner, etc...). From a software design standpoint, the knowledge "is declarative in that it presents the logic of our analysis and procedural in that it is, in fact, a component of the logic program itself". This illustrates the interstitial role of knowledge as orchestrating the logic of the activity.

3. Weiskopf et al. (1990) have integrated the knowledge component (strategic level) with neural net routines to deal with soft constraints. The knowledge level is the executive level, controlling the execution, while the neural nets are the controlled routines.

The cognitive principles have a direct impact on the knowledge acquisition process. In the case of atomic knowledge, it is essential that knowledge can be rapidly transferred, from a given situation or observation into a local formalism In the extreme case of a purpose specific control structure, more monolithic knowledge acquisition tools are to be used. What links the two extremes is the software engineering which requires that both types of knowledge can co-exist in any given application.

Acquiring atomic knowledge

At the atomic level, it is necessary to be able to express any chunk of knowledge and express the relations between them. Tools that have the correct cognitive characteristics. i.e. visual interaction, simplicity and clarity of expression and representation (Rappaport 1988, Gaines 1991, Shaw and Gaines 1991). With the availability of such tools and the software engineering integration resolved to a satisfactory extent, the relation of power to expressibility (Gruber 1989) will be modified. The power is redefined according to the match to the user's perspective, not the performance of a language.

For example, a user of a GIS software as exemplified above. The user may be making scenarios on disaster assessment, designing new transit routes, marketing products or diagnosing a wide-area network. As it becomes possible for such users to access a common set of knowledge (cognitive) primitives and concept, he/she must be able to concurrently formalize, in relation to the tools they are using, the relationships and/or rules which will be useful in future interactions. Direct interactive implementation of the local knowledge enriches the environment. Interstitial knowledge links the environment libraries to others in the context of the activity.

Acquiring agenda-based knowledge

The agenda structure maps into the notion of focus of attention. The semantics of the activity as a management of subactivities is the product of this knowledge acquisition process. Cognitive

primitives allow for this type of interactions designed to map the trajectory of the activity.

Specific knowledge

The acquisition of specific knowledge for hardwired control structures remains an important issue. The critical point to emphasize is the architectural design of such tools so it can (i) integrate and (ii) communicate or be executed according to interstitial knowledge and abstract data structures.

Acquisition Infrastructure

The acquisition of local and interstitial knowledge requires that a number of supporting techniques constitute a stable, activity-independent infrastructure. The latter must adequately support the important characteristics of local knowledge such as opportunicity. If local knowledge is opportunistic, the acquisition of the same knowledge is opportunistic as well. If it is agenda-based, event-driven, then the knowledge acquisition process must be also event-driven, expressing the way the activity unfolds.

Such an infrastructure includes re-useable widgets to graphically express and edit knowledge, (ii) indexing mechanisms to easily retrieve, display, edit knowledge structures, (iii) mapping into traditional repositories (relational, hierarchical, object-oriented as needed), (iv) parametrized truth maintenance systems to detect inconsistencies at desired levels, (v) knowledge servers and distributed environments, (vi) automatic knowledge generation and hyper-documentation. Although this list is partial, it emphasizes the need for re-useable support libraries in order to implement local and interstitial knowledge in a transparent way. The user-interface widgets are fundamental in that they constitute the most visual and interactive metaphors with which to edit or create knowledge. In the GIS example (#1 above), the map is the fundamental metaphor and the knowledge acquisition widgets are likely to be extensions of the basic domain primitives. The same would happen in a molecular biology situation where there are different but also basic domain-specific metaphors and representations.

Background acquisition

Induction algorithms and other machine learning technqiues are 'backgound' activities. They must be architectured to integrate with the universe described so far, having rapid and simple access to data and generating a useable formalism in any given system. With respect to the cognitive principles described earlier, these techniques 'renormalize' the information so it can be used by a process with a focus of attention. The backgound activity is also where weak problem solving techniques are progressively transformed into stronger ones, more activity-specific as in Boy, 1990.

The management of the knowledge level in this perspective requires itself the use of local knowledge. In any given instance, it is by interacting with the user of the system (the system that makes use of knowledge) that the proper libraries will be invoked to create the acquisition process.

Methodologies can be developed for such a knowledge acquisition process but it would differ somewhat from those designed to help the knowledge engineer (Hayward et al. 1988). New approaches are closer to our preoccupation (Motta et al., 1991). Most critical is the need to tightly integrate again the knowledge acquisition tools with the performance environment (Rappaport and Gaines 1990), knowing that the tools in question might well be more traditional techniques or non-

AI data processing techniques integrated with the infrastructure described above. Knowledge is now not only a matter of knowledge engineer but also very much a matter of software and application design. A smaller percent of the knowledge in a functioning integrated architecture is often better than a larger percent of the knowledge turned irrelevant because of a closed design or a restrictive formalism ill-adapted to the problem.

LOCALITY AND THE EMERGENCE OF SEMANTICS

A first guess at a characteristic of local knowledge is that it would be non orderly and perhaps too simple and inefficient. Openness means more uncertainty than closed spaces. However, closed spaces do not work because the reality of human activities cannot be thus formalized in terms of tasks or algorithms. Local knowledge processing transcends the classic differences in determinism between symbolic systems and non-symbolic architectures because its ubiquitous implementation forces in all cases a risk-management model.

By allowing the human interactions to be rapidly transcribed into chunks of knowledge, which may or may not be used and need to be revised (like humans need to revise their own knowledge), into the computing infrastructures we basically introduce the same type of social and economic factors as those involved in hiring skilled individuals. The computer becomes a true extension of the local mind, i.e. the processing in context. Inasmuch as human activities have a non-linear trajectory, local knowledge is better adapted to enhance these activities.

Regardless of the determinism of the rules or other representations involved, local knowledge there is introduction of an element of risk beyond the simple presentation of data. However, the trade-off is largely in favor of the fact that local knowledge brings back the semantics of the programs, thereby linking them constructively and efficiently to the real-world problem being addressed. Monolithic AI systems have never given the opportunity to observe this because by definition they do not "mingle" with the rest of the infrastructure and thus their theoretical benefits cannot be practically observed. By adding local and interstitial knowledge processing we will be able to rapidly bridge the gap between the low level functions performed by the computers today and the more abstract, higher level considerations that make us use those very computers for all purposes.

PERSPECTIVES

The theory of local knowledge advocates that even small amounts of knowledge can transform a process by integrating it better to the broader context and thereby making it more efficient to the user, be it a human or another computing process. We speak of 'injecting' knowledge processing in existing and future architecture. Furthermore, the technology must be scalable, nucleating from small amounts of injected knowledge within an architecture to being actually run and designed essentially at the knowledge level. That level is 'super-object-oriented', manipulating objects as ACDS and entire libraries as an executive process. As said before, locality is defined by time and acitivity. Much local knowledge is common to many activities and should be shared. The present theory is very general. Work has been in progress on common-sense background knowledge bases or servers (Lenat and Guha 1990), in non-standard software environments so far, in distributed problem-solving (Pan and Tennenbaum 1991) or reconfigurable architectures (Gasser and Ishida, 1991).

If most systems could perform not only at the process level but also at the knowledge level, the gap between the actual purpose of running the processes and the overall objective can be filled much quicker. We are working on formalizing many of the aspects of the local knowledge theory from the articulation of problem solving approaches (locality, distribution) to the overall economic implications (economy of processing, economy of adaptation and other economies). The interdependence of the approach with fundamental software engineering issues (architecture, execution modes) make it particularly important to further develop this research. Knowledge acquisition and processing is thus becoming part of the way we write software or develop applications. Soon enough it should also fundamentally influence those activities and the way we design languages, tools, operating system and future software.

Acknowledgements: Thanks to many at Neuron Data. Special thanks to Jason Catlett, Jean-Marie Chauvet and the reviewers for their comments on this paper, to Brian Gaines, Raj Reddy and others for valuable discussions.

REFERENCES

Averback, J.M. (1991) The Human Genome Initiative, *Iris Universe*, 35-39.

Boehm B. (1988) A Spiral Model of Software Development and Enhancement, *Computer*, 20:61-72.

Boy G (1990) An Intelligent Assistant for Acquiring Procedures, *Proceedings of the AAAI-90 Workshop on Knowledge Acquisition*, Boston, MA.

Catlett, J. (1991) Megainduction: a Test Flight, *Proceedings of the 1991 Machine Learning Workshop*, Morgan Kauffman, San Mateo, CA.

Chandrasekaran B. (1989) Task-Structures, Knowledge Acquisition and Learning, *Machine Learning*, 4:339-345

Compton P. and Preston P. (1990) A Minimal Context Based Knowledge Acquisition System, *Proceedings of the AAAI-90 Workshop on Knowledge Acquisition*, Boston, MA.

Erlich E. et al. (1980), Oxford American Dictionary

Gaines, B. R. (1989) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data Based on Statistically Well-Founded Empirical Induction, *Proceedings ot the Sixth Machine Learning Conference*, Morgan Kauffman, San Mateo CA, 156-159.

Gaines, B. R. (1990) Knowlegde Acquisition Based on an Open-Architecture Knowledge Representation Server, *Proceedings of the AAAI-90 Workshop on Knowledge Acquisition*, Boston, MA.

Gaines, B. R. (1991) An Interactive Visual Language for Term Subsumption Languages, in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Morgan kauffman, San Mateo, CA.

Gasser L. and Ishida T. (1991) A Dynamic Organizational Architecture for Adaptive Problem Solving, *Proceedings of the Ninth National Conference on Artificial Intelligence*, Morgan kauffman, San Mateo, CA, 185-190.

Hayward S.A. et al. (1988) Structured Analysis of Knowledge, in Knowledge-based Systems, Academic Press, London, UK.

Grashion A. R. (1990) The Use of Lotus Agenda in Knowledge Acquisition for an Expert System in a Poorly Controlled Domain, *Proceedings of the Fifth Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada.

Kotokovski K. and Simon H.A. (1989) What Makes Some Problems Hard: Explorations in the Problem Space of Difficulty, Carnegie Mellon University, TR-AIP-97.

Djokic D. and Maidment D et al. (1990) Terrain Analysis For Urban Stormwater Modelling, Hydrological Processes, 5:115-124.

Lenat D.B. and R.V. Guha (1989) Building Large Knowledge-based Systems, Addison-Wesley.

Meyer B. (1988) Object-Oriented Software Construction, Prentice Hall, New York, NY.

Motta E. et al. (1991) Methodological Foundations of KEATS, the Knowledge Engineer's Assistant, *Knowledge Acquisition,* 3:21-47

Pan J.Y-C. and Tennenbaum M. (1991) Toward an Intelligent Framework for Enterprise Integration, *Proceedings of the Ninth National Conference on Artificial Intelligence*, Morgan kauffman, San Mateo, CA, 206-211.

Rappaport A. (1988), Cognitive Primitives, *International Journal of Man-Machine Studies*, 26:733-747

Rappaport A. and Gaines B. (1990) Integrated Knowledge-base Building Environments, *Knowledge Acquisition*, 1: 51-73.

Rappaport A. (1990), Communication at the *Japanese Knowledge Acquisition Workshop,* Kyoto, Japan.

Shaw M. L. and Gaines B. R. (1991) Using Knowledge Acquisition Tools to Support Creative Processes in *Proceedings of the AAAI-91 Knowledge Acquisition Workshop for Knowledge-based Systems*, Banff, Canada.

Simon H.A. and Lea G. (1974) Problem Solving and Rule Induction, in Models of Thoughts.

Yale University Press, New Haven, 1979.

Simon H.A. (1981) Sciences of the Artificial, MIT Press, Cambridge, MA.

Simon H.A. and Feigenbaum E. (1984) EPAM-like Models of Recognition and Learning, *Cognitive Science*, 8: 305-336.

Sloane, S.B. (1991) The Use of Artificial Intelligence by the United States Navy: Case Study of a Failure, *AI Magazine*, 12:80-92.

Weiskopf F.B. et al. (1990) A Hybrid System Approach to Machinery Monitoring and Diagnosis, in *Proceedings of the Second International Machinery Monitoring and Diagnosis Conference*, Los Angeles.

On the Limitations of Repertory Grids in Knowledge Acquisition

Gordon Rugg and Nigel R. Shadbolt
Artificial Intelligence Group
Department of Psychology
University of Nottingham
Nottingham NG7 2RD England
nrs@psyc.nott.uk.ac

Results obtained using repertory grid techniques and tools can fail to properly represent significant aspects of expert knowledge. This is a growing problem as the technique becomes more widely disseminated and the users less specialist in the background theory of grids. We review where problems can arise in the construction and analysis of grid data. We illustrate a number of particular difficulties in the context of an extended case study in the domain of mineralogy. The case study reveals situations where knowledge is inappropriately or inadequately characterised. The case study is not a large scale empirical investigation. It is only intended to be indicative of types of problem that may arise. A number of suggestions are made for ameliorating these difficulties.

INTRODUCTION

Repertory grids are well established in knowledge acquisition (KA), and a variety of software implementations of the repertory grid technique are available (Boose, 1990; Boose and Bradshaw, 1988; Shaw and Gaines, 1988).

It is not surprising that repertory grids should be so popular for KA. They have numerous attractive features, such as standardisation of data format and ease of automation. It is easy to pool knowledge from multiple experts and to use statistical analyses to bring to light underlying patterns in the data.

Repertory grids, used properly, are a powerful tool. Unfortunately, their apparent utility is deceptive. The technique rests on a variety of critical choices, decisions and assumptions. Each of these issues introduces the possibility of error or distortion. These problems are fairly well known in psychology, where the technique originated (cf for example, Bannister and Mair, 1968). They are not so well known to the knowledge engineer who is using the technique in the context of KA. There are no suitable write ups which alert the knowledge engineer to the problems of grid construction and analysis. This paper attempts to draw these potential problems to the attention of the knowledge engineer. We would not want to discourage the use of the technique. Indeed we advocate its inclusion within KA toolkits (cf for example, Reichgelt and Shadbolt, 1991). We would, however, urge that the technique be used *knowledgeably*.

To make the relevant issues clearer, a worked example or case study will be used. The case study involves the description and classification of minerals in geology. The domain is clearly defined and involves knowledge which is well established and theoretically sophisticated.

The first part of the paper discusses how grids are constructed and analysed. We discuss some of the issues raised at the outset of grid construction. We then provide an account of our case study. The case study comprises two parts. In the first part, a manual grid was used to elicit knowledge from an expert in mineralogy. Although almost never used in knowledge elicitation in this manual way, we chose an initial pen and paper approach to avoid, at this stage, various implementation dependent issues. On finishing the grid the expert was asked to express his satisfaction with the representation of his knowledge within the constraints of the repertory grid format. The expert was then interviewed to determine the constructs that he would use to describe and classify the chosen subset of minerals. In the second part of the case study, a computer implementation of a repertory grid was used and the grid elicited was analysed automatically. The results were then fed back to the subject for comments. Finally, we review conclusions and suggest ways in which the repertory grid technique can be further refined and developed for use in the context of KA.

THE REPERTORY GRID TECHNIQUE

In using repertory grids as a KA technique we are relying on the ability of the method to reveal an expert's conception of his domain of expertise.

> A repertory grid is a two-way classification of data in which events are interlaced with abstractions in such a way as to express part of a person's system of cross-reference between his personal observations or experiences of the world (elements) and his personal constructs or classification of that experience. Shaw and Gaines, 1986.

The idea is that humans organise the world in terms of set of dimensions called *constructs*. So in mineralogy we might have a construct *common–rare* along which we can classify minerals we are familiar with. Another one might be *radioactive–nonradioactive*. The expert's view of the domain will organise elements along these various constructs.

In building a repertory grid the constructs are understood to possess a range of values with each extreme end of the range being termed a *construct pole*. The elements, constructs, and construct poles are all usually given linguistic labels. In the manual version the grid usually begins with only the elements filled in, and the session is used to elicit constructs and cell values. Usually, the constructs are elicited by systematic comparison of sets of three elements (triads), in which the expert decides what construct pole is shared by two of the triad but not the third. All of the elements are then given numeric values for the extent to which they exhibit the construct, filling in the cells of that line of the grid. The procedure is then repeated for the next lines of the grid, until the end of the session.

When the elicitation of constructs is complete a statistical procedure called *cluster analysis* can be performed on the data. This procedure groups constructs which are similar in the sense that elements are rated similarly on them. The results are presented in a graphical structure called a *dendrogram*. The same procedure is applied to the elements. Again we can observe which elements of the domain are grouped together.

The most basic application of this technique immediately raises a host of issues regarding grid construction and analysis (cf for example Yorke, 1985). We will raise them here before proceeding to our case study.

ISSUES IN GRID CONSTRUCTION

Choosing elements

One of the first decisions in constructing a grid is the choice of elements. The most obvious kind of element is one that stands for a concrete object in the domain, such as a particular mineral. It is likely that constructs will be elicited that are properties of these concrete objects. The technique in no way predetermines the type of thing that could act as a domain element. Elements can stand for any component of knowledge. For example, the elements could just as well be tasks or goals.

However a basic requirement is that the elements in any one grid should be of the same epistemological type and at the same level as each other. One should not mix instances of objects, with classes, goals, tasks etc.

Assuming that one is consistent then there is the issue of representativeness. How are we to ensure that the element choices are representative for the purposes of acquisition? We have to take care lest the element set chosen be skewed or unrepresentative in various ways (Yorke, 1985). Considerable knowledge of the domain may be needed to select a suitable set.

An aspect of representativeness involves semantic differentiation between the elements. If the elements are too homogeneous, then they may be indistinguishable except in terms of obscure or trivial descriptive dimensions, and important common dimensions may not be elicited. Conversely, if the elements are too heterogeneous, then it may be difficult applying any but the broadest dimensions to them. If there is a mixture of some very heterogeneous and some very homogeneous elements then the situation becomes problematic, and the choice of elicitation triads may be critical.

The issue of semantic differentiation is related to the concept developed in Personal Construct Theory of "range of convenience". This is the extent to which the constructs differentiate elements. Constructs with a low range of convenience are ones that can be used meaningfully in a narrow range of contexts. The converse holds for constructs which differentiate elements with a high range of convenience.

Measurement

Assuming that the choice of elements has taken place without problems, another set of choices has to be made about allowable cell values in the grid. A wide range of options is available.

The most basic type of grid allows only two values, namely 'yes' and 'no' (or 'present' and 'absent'). This indicates the acceptable ascription of the construct to the element. A more sophisticated type allows values on a rating range of, for example, one to five or one to seven. Another option is to use a ranking system, from 'most' to 'least'. When rating or ranking is used, the knowledge engineer has to decide whether or not to

allow tied values (e.g. two elements receiving a 'three' rating or ranking on the same dimension). There is also the option of allowing the user to choose the type of values they want to use for each dimension, so that the finished grid may contain a mixture of ranking and rating comparisons. The knowledge engineer may want to include a 'not applicable' value and/or a 'don't know' one. As far as the matrix itself is concerned, blank cells present no problems — indeed, they may be informative. A difficulty arises when one wishes to subject the matrix to statistical analysis. Enjoining the subject to use the mid-point whenever a construct cannot subsume an element is not always a sensible or viable option, particularly in ranking.

The decision made at this point constrains the type of statistical analysis that can be used. Broadly speaking, the most powerful statistics can be used only where there are no tied values, no 'don't know' and 'not applicable' values, and with no mixture of ranking or rating in the same grid.

Implicit in the options above is the assumption that the dimension used can be described either in terms of 'yes/no' binary values, or in terms of a simple unilinear scale.

Considerable attention has been paid to these issues in psychological research where the repertory grid technique has its origins. Thus it has long been recognised that rated grids and the assumption of unilinear scales are not without their problems. For example, the assumption of metrical equivalence between constructs — this is the assumption that the same distance on two constructs can be treated as similar semantic distances in the conceptualisation of the expert. Yorke (1983) showed that such assumptions are unlikely to be warranted. Other researchers (Gaines and Shaw 1981; Thomas and Shaw, 1976) have all noted problems in the measurement assumptions made by grids. We will return to these issues later.

Choosing constructs

Finding constructs is usually one of the most important tasks in using a repertory grid, and will be used as the main part of the worked example.

One initial question is whether the knowledge engineer actually wants to elicit constructs. There are situations where it may be useful to supply both the elements and the constructs, and use the expert to fill in the cell values. This is most often the case when pooling or comparing knowledge from different experts: use of supplied elements and constructs makes such a socio grid approach possible (Shaw and Gaines, op.cit.).

If the constructs are to be elicited from the expert, then a wide range of issues become salient as the case study illustrates.

As well as allowing the expert to volunteer constructs we noted earlier that triadic elicitation is often used in construct elicitation. Even here there is debate. It is argued that diadic elicitation (using pairs of elements) is intrinsically more stable that the triadic method (Yorke, 1978).

A crucial problem is understanding fully the intended semantics of the construct which the subject is using. Many psychologists urge the use of interview techniques to supplement knowledge about the meaning of constructs. It is likely that a construct and its various linguistic labels are associated with considerable amounts of contextualised knowledge.

The construct's linguistic labels will usually be highly polysemous — carrying for the expert many assumptions, associations, implications, caveats, qualifications and the like. Some method is needed to indicate the sense in which the construct is being construed during a grid session.

Having highlighted some of the basic issues of grid construction let us now move on to our case study.

CASE STUDY: PART I THE MANUAL GRID AND INTERVIEW

Method

A manual repertory grid was used to elicit mineralogical constructs and values from an experienced geologist with specialist knowledge of mineralogy. The elements were minerals found in igneous rocks (i.e. domain objects). The cell values were rankings, with non tied ranks. The expert was introduced to the background theory of repertory grid practice before the elicitation. The expert's comments on the constructs were noted during the session. After all the constructs had been elicited, the expert was asked to rate them in terms of distortion by the knowledge representation, as either acceptable, somewhat unsatisfactory or very unsatisfactory. This rating reflected the extent to which the cell values in the grid differed from the values which the expert would have preferred to use, due to the constraints of the grid type being used.

Immediately after the grid had been completed, an interview was used to elicit the constructs which the expert might normally use to classify and describe the world of igneous rocks. The expert was then given the opportunity to add any of these to the repertory grid (in case they had overlooked any during the grid session). Finally, the subject was asked to give a description of plagioclase feldspar (one of the elements used) in his usual way.

In mineralogy, as in most expert domains, there are disagreements between experts about the usefulness and optimal usage of domain concepts: textbooks vary in the number and values of dimensions used to describe minerals. The descriptions elicited in this case study should therefore be treated as illustrative, rather than definitive statements about mineralogy

Results

The raw grid elicited is shown in Figure 1 and shows the constructs and ranking provided by the expert. Blank entries are where the expert was unwilling to make a ranking judgment at all. The constructs which were elicited are enumerated again in Table 1. The first set represented in the first three rows of Table 1 are those that can be seen in hand specimen (using a hand lens or unaided visual inspection. The second set of constructs, rows four to ten of Table 1, are those associated with examining the mineral in a thin section. Thin sections can be viewed using a petrological microscope.

	plagioclase	k_feldspar	olivine	garnet	muscovite	biotite	hornblende	quartz
ts-hi-relief/ts-lo-relief			7	8	4	5	6	
ts-multi-twin/ts-no-twin	8	7					6	
ts-coloured/ts-colourless	3	2	6	1	5	8	7	4
ts-cleavage-present/ts-clevage-absent					6	7	8	
ts-alters-easy/ts-alters-no	8	7	5			6	4	3
ts-hi-biref/ts-lo/biref			6	8	7	4	5	
hs-hard/hs-soft	5	6	7	4	1	2	3	8
hs-specgrav-hi/hs-specgrav-lo	2	3	7	8	4	5	6	1
ts-simple-shape/ts-complex-shape	4	3	1	8	5	6	7	2
hs-coloured/hs-colourless	2	3	7	8	5	6	4	1

Figure 1 The raw grid as elicited from the expert

The third column of Table 1 illustrates the expert's assessment of how well represented the construct was within the constraints of the grid. As we can see, the constructs which the subject regarded as being represented acceptably were colour in hand specimen, hardness and specific gravity. The constructs which the subject regarded as being represented with some distortion were colour in thin section, alteration, birefringence, and grain shape. The constructs which the subject regarded as being represented with a lot of distortion were twinning, cleavage and relief.

Associated with	Construct	Satisfaction
Hand specimen	Colour	Acceptable
Hand specimen	Hardness	Acceptable
Hand specimen	Specific gravity	Acceptable
Thin section	Twinning	Large distortion
Thin section	Colour	Some distortion
Thin section	Cleavage	Large distortion
Thin section	Relief	Large distortion
Thin section	Alteration	Some distortion
Thin section	Birefringence	Some distortion
Thin section	Grain Shape	Some distortion

Table 1 Expert's assessment of construct distortion using the ranking method

The constructs elicited using the interview are shown in Table 2. Those constructs annotated with a ‡ were regarded as of only marginal usefulness.

Associated with	Construct	Present in repertory grid
Hand specimen	Colour	Yes
Hand specimen	Hardness	Yes
Hand specimen	Specific gravity	Yes
Hand specimen	Lustre	No
Hand specimen	Grain Shape	No
Hand specimen	Form	No
Hand specimen	Taste‡	No
Hand specimen	Odour‡	No
Hand specimen	Cleavage	No
Hand specimen	Fracture	No
Hand specimen	Tenacity	No
Hand specimen	Magnetism	No
Hand specimen	Radioactivity	No
Thin section	Twinning	Yes
Thin section	Colour	Yes
Thin section	Cleavage	Yes
Thin section	Relief	Yes
Thin section	Alteration	Yes
Thin section	Birefringence	Yes
Thin section	Grain Shape	Yes
Thin section	Extinction	No

Table 2 Constructs elicited from the expert during interview

Discussion of results

The most immediately striking result is that the interview elicited twice as many constructs as the repertory grid.

The second feature is the low number of constructs which the expert regarded as satisfactorily handled by the repertory grid: only three constructs out of ten, with those ten in turn forming only a subset of the constructs which the expert would normally use. As we will discuss even these three "satisfactory" constructs are not unproblematic. Moreover, in a number of instances the expert would not rank the elements at all. Thus the grid failed to represent in a completely satisfactory way any of the expert's domain knowledge.

One of the constructs judged satisfactory by the expert was colour in hand specimen. However, subsequent discussion with the expert made it clear that colour is actually a complex issue in mineralogy. ''Colour'' can refer to either the strength of a colour (which can be represented fairly easily on a repertory grid) or the actual value of the colour (e.g. red, green or blue), which is much harder to represent on a repertory grid.

Even with strength of colour, there is the problem that some minerals may vary in colour and/or strength of colour: quartz is a particularly good example of this, and may even be colourless.

The two other constructs judged satisfactory, namely hardness and specific gravity, are dimensions which in practice have numbers as values. However, problems can arise even in these cases. For both "hardness" and "specific gravity" the values used in mineralogy are absolute values (like ratings) rather than relative ones (like rankings). The ranking value for "hardness" for plagioclase feldspar, for instance, was given by the subject as 5 (on an eight point scale). However, the standard value for plagioclase's hardness is about 6 on the Moh scale of hardness. In these circumstances, with established absolute values readily available, a rating system allowing both tied values and continuous numeric scales would be the most appropriate reflection of the expert's knowledge.

We have already alluded in the section on grid construction to the problem of assuming simple Euclidean metric space for the semantics of a construct. Returning to Mohs' scale of hardness we discovered that in this scale the intervals between points on the scale are not all equal. Attempts to map a scale with irregular intervals onto one with regular intervals could play havoc with incautiously applied statistics used to investigate perceived similarity of elements.

The constructs judged moderately unsatisfactory were colour in thin section, alteration, birefringence and grain shape.

Colour in thin section was used in the sense of how strongly coloured the mineral was. The expert was unhappy with this construct because it was not possible to represent within the grid how much variance there could be within the same mineral.

The construct "alteration" caused problems because some minerals don't alter, which meant that without tied values many of the rankings were essentially arbitrary. Ideally, the minerals which do not alter should have all received a tied value of zero, but instead they all had to be given positive values different from each other. In fact, as we have already noted the expert on a number of occasions simply declined to rank the elements. Birefringence caused similar problems, since quartz had to be arbitrarily differentiated from the feldspars purely as a result of the requirement for untied values.

The problems with "grain shape" occurred because the shapes involved are discrete and qualitatively distinct categories. The expert was reduced to ranking the number of sides in a grain of each mineral, instead of describing the actual shape.

The constructs judged very unsatisfactory were twinning, cleavage, and relief.

The problems with "twinning" arose because twinning subdivides into discrete classes rather than forming a continuum. Some minerals show multiple twinning, for example, whereas others may show simple twinning, some simple twinning, or no twinning.

The mineralogical construct "cleavage" caused difficulties because the expert used the construct as a simple bipolar one of whether or not cleavage was present. This meant that the rankings used were essentially arbitrary.

With "relief", half of the elements had low relief, and would require more detailed study to be distinguished petrographically from each other. A more serious problem is that relief refers to the refractive index of the mineral, for which an absolute value can

be given. This means that the construct suffers both from largely arbitrary values, and from trying to use a ranking system when absolute values are available.

An issue common to many of the dimensions is that the values specified may be a range, e.g. a mineral may have a range of values, say 2.6 - 2.7, for a construct such as specific gravity. This is not normally a problem for individual specimens, where an exact value can usually be ascertained, but it is a problem for general descriptions, since different examples of a single mineral may exhibit a range of values for a property or construct.

In the first part of our case study we restricted ourselves to a ranking format. Nevertheless many of the problems we are raising would persist if our format was to change. The representation system in repertory grids is one of attribute and quantifier, rather than attribute and value. In other words, a repertory grid records how much of something is present. The value may be a ranking between the element which displays the highest amount of the construct and the element displaying least of it; it may be a rating measuring how much or how little of the construct is present; it may be a bipolar value measuring simply whether the construct is present or absent. The essential point is that the cell entry represents a measurement of something which has a range of possible values.

This representation itself leads to some severe problems in knowledge elicitation. In the case of mineralogy, it affects at least six of the ten dimensions elicited in the manual grid. The most common problem occurs when there is a well defined set of categories which cannot be represented as a set of quantitative values.

One example of such a set is colour terms, which are relevant to both colour and streak in mineralogy. (Streak was not mentioned by the subject because none of the minerals involved would give a streak.) Colour is usually described in terms of colour words, such as 'green' or 'greyish white'. These words provide a quick, convenient way of describing colour for everyday purposes, and are adequate for many technical purposes as well. However, they do not represent quantities of a dimension. Asking: 'how much colour does X have?' is quite a different question from asking: 'what colour is X?' The second question will produce an answer which is a colour name; the first may provide a variety of types of answer, depending on context.

In the case of colour it is possible, though cumbersome, to find ways of representing values within the framework of a standard repertory grid. One way is to use bipolar 'present or absent' values for each colour in turn, so one line of the grid records whether or not each element is green, for example, while the next line records whether or not each is yellow, and so on. This would clearly be extremely clumsy and inefficient. Another possible solution is to use a physics-based approach, and record the hue, saturation and so forth corresponding to each colour. This would require several lines of the grid for each colour, and would require a certain amount of technical knowledge in its own right.

In other cases, however, such a solution is not possible. Such cases involve dimensions split into nominal categories which do not form any sort of scale, and do not blend into each other. Thus, for example, the dimension "propulsion type" for vehicles might include the categories "internal combustion", "wind" and "electricity". These are clearly members of the same higher level group, but they do not form a continuum: "internal combustion" does not blend into "electricity" in the way that "red" blends

imperceptibly into "orange" in the dimension of colour. In mineralogy, categorical distinctions affect constructs such as "forms", "lustre", "cleavage" and "fracture".

Such categories can obviously be very useful in describing a domain, and may be very formally defined and well agreed within a domain: however, there is simply no way of representing them in a standard repertory grid other than with bipolar values for the presence or absence of each category. This is not only inefficient, but also loses the semantic relationship between related categories: there is nothing to show that one category is an alternative to a second, but orthogonal to a third.

The constructs elicited in the interview, but not in the grid, were: extinction, lustre, grain shape in hand specimen, form, taste (marginal), odour (marginal), cleavage in hand specimen, fracture, tenacity, magnetism, and radioactivity. The expert when asked if he wanted to add the constructs from the interview to the repertory grid declined to do so on the grounds that the ranking method distort the constructs. Notwithstanding, discussion of these will focus on those constructs which raise fresh points of interest in relation to knowledge representation in repertory grids.

One construct which raises such issues is "form". The element plagioclase may take any of several common forms. This is an extension of the issue of a range of values discussed above. There is no convenient way of representing such alternative values in a standard repertory grid: as usual, it may be possible to use bipolar values for the presence or absence of each alternative form, and to describe the same mineral once for each alternative on a different line on the grid, but this is a somewhat cumbersome solution.

The construct "twinning" raises problems related to *range of convenience* and to which we have already referred. Constructs with a low range of convenience are ones which can only be used meaningfully in a narrow range of contexts: those with a high range of convenience can be used in a wide range of contexts.

Attributes with a low range of convenience may be extremely useful: for instance, plagioclase feldspar may display an unusual feature known as 'multiple twinning', which identifies plagioclase with a very high degree of certainty, if present. This was not in fact picked up in the manual grid. It is possible that constructs with a low range of convenience may be elicited less often than constructs with a wider range of convenience. The latter are often useful in a wider range of contexts than the former, and there is evidence that experts tend to make more use of wide ranged constructs than do non experts (Murphy & Wright, 1984).

Having reviewed some of our findings and their implications for the grid user we will now move on to the second part of our case study.

CASE STUDY: PART II AUTOMATIC ANALYSIS

Method

After the manual construction of a grid and the interview which we have described in the previous sections we conducted a second session in which the same subject, with the same elements and constructs as in the previous session, used an implemented repertory grid

tool (Reichgelt and Shadbolt, 1991). This used a ten-point rating system with tied values allowed. When the grid had been completed, it was subjected to automatic analysis and the results from this were fed back to the expert for comments. The automatic analysis methods are based on the use of similarity matrices derived from element and construct distances computed using a 'city block' metric, (cf for example Shaw, 1980), together with the induction of implications between constructs (Gaines and Shaw, 1986).

The results used for the feedback were the dendrograms of similarity for the constructs and elements, and the implicative rules for constructs. The dendrograms were standard ones, showing the clusterings of similarities which were found statistically between minerals and constructs respectively. The implicative rules were of the form *construct1 implies construct2*; for example, *''coloured in thin section'' implies ''cleavage present in thin section''*[1].

The subject went through the thirty-five implicative rules which exceeded a threshold of 67% true (the algorithm assigned 8 rules as 100% true, 16 as 89%, 2 as 78% and 9 as 67%). The expert judged the extent to which each of these rules was true for the domain in general as opposed to the test set (the domain being minerals likely to occur in igneous rocks). After this, the eight rules with 100% truth value scores were assessed in terms of whether there was any genuine (rather than accidental) link between the two constructs involved in each rule. Generalisable truth was assessed using the three categories of ''true'', ''false'' and ''neither''. The genuineness of links between constructs was described verbally, as were the dendrograms.

Results

We first present in Figure 2 the rating grid which the expert produced. The expert was told that in this case the grid must be complete, and that a rating must be given for each element and construct[2].

---

[1] The program supplied associated ''truth'' figures for the percentage of relevant cases in the sample where this relationship applied, and for the number of relevant cases, but these were not used in the feedback to the subject.

[2] Note that the directionality of the rating in Figure 2 is not necessarily the same as for the ranking score in Figure 1.

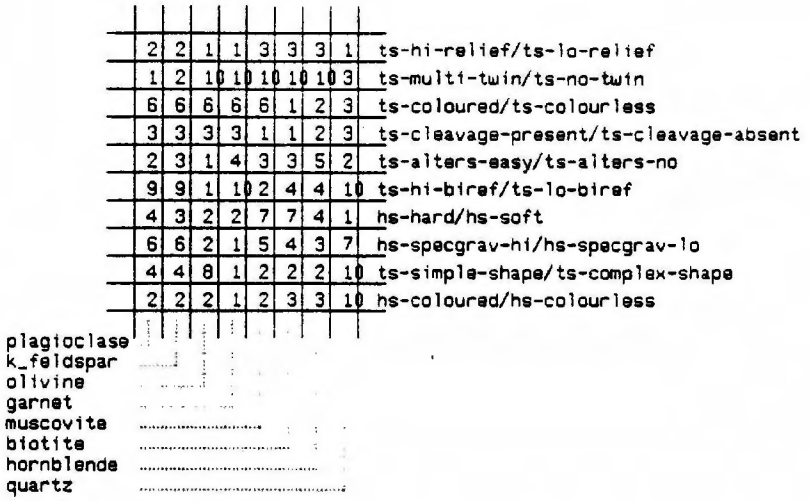


Figure 2 The rating grid for TACKTIX as elicited from the expert

Figures 3 and 4 are the element and construct dendrograms generated by the computer program on the basis of the grid given in Figure 2

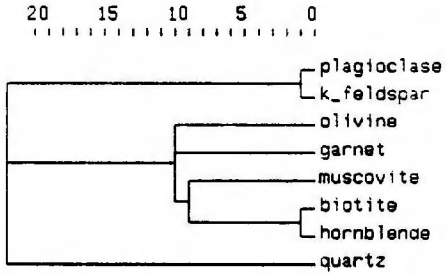


Figure 3 Element dendrogram generated by the program

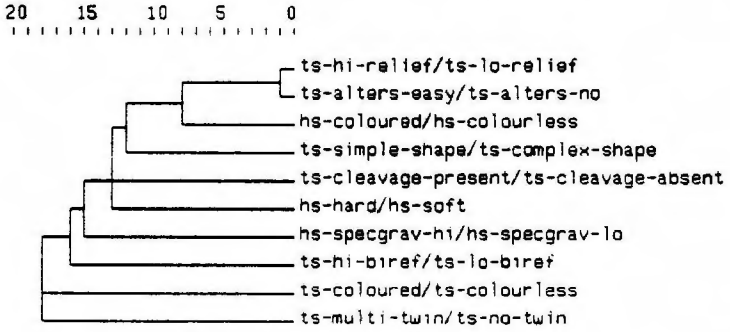


Figure 4 Construct dendrogram generated by the program

Table 3 shows the expert's assessment of the four highest categories of implication rule generated by the system. Table 4 shows a more detailed analysis of those implications that the system considered to be 100% true.

System Score	True	False	Neither	Total
100	5	2	1	8
89	4	11	1	16
78	0	2	0	2
67	3	3	3	9
Total	12	18	5	35

Table 3 Expert's assessment of implications generated by the program

Implication	Generally true/false/na
Thin section coloured implies thin section cleavage present	False
Thin section alters easily thin section high relief	False
Hand section hard implies thin section high relief	True
Thin section low relief implies hand section soft	True
Thin section low relief implies thin section doesn't alter	Neither
Thin section cleavage absent implies thin section colourless	False
Hand section specific gravity high implies thin section high relief	True
Thin section low relief implies hand section specific gravity low	True

Table 4 Some of the implications generated by the automatic repertory grid tool

Discussion of the results

In Figure 3 the mineral dendrogram, the expert judged the two feldspars (orthoclase or K_feldspar and plagioclase) to be similar to each other, which agreed with the dendrogram, and judged muscovite and biotite to be similar to each other, which was close to what the dendrogram showed (the dendrogram shows a higher cluster between biotite and hornblende, which the expert judged to be spurious). Otherwise, the dendrogram was judged to be misleading as a reflection of the similarities between minerals. The expert judged olivine and garnet, quartz and the feldspars as similar minerals —clearly at odds with the picture implied by the dendrogram.

With the construct dendrogram, Figure 4, the expert initially had some difficulty in believing that the constructs were actually being associated with each other in the way shown in the dendrogram. The link between relief in high section and alteration was judged to be valid; one between hardness and specific gravity was notable by its absence. The rest of the links were judged to be very dubious.

Only about a third of the implicative rules in Table 3 were judged to be more than coincidental associations; similarly (but separately), only about a third of these rules were judged to generalise to the rest of the domain from the test set. It should be stressed at this point that the minerals used as the elements in the grid were not unusual, atypical ones: on the contrary, they are very common minerals which have at least as much claim as any other mineral to be considered "typical".

One might expect a certain amount of noise creeping in with the rules scoring lowest on truth values, but even with the rules in Table 4 which scored highest on this, five of the eight were judged to be meaningless, accidental associations. The rule which scored highest on both truth and number of relevant cases (100% for both) was judged to be both untrue for the domain in general and to be a spurious association as well.

One last point which should be made about automatic computerised repertory grids, involves the use of real-time iterative statistical analysis to identify elements which have not been differentiated. The program can then ask the subject for a construct which does differentiate the relevant elements. Although this approach clearly has its uses in some situations, it can impose a greater degree of differentiation, and greater heteroscedacity, than the subject wanted to use.

Medicine provides a good example of this point. The concept of differential diagnosis involves knowledge of disorders sufficiently similar to a given disorder to be easily confused with it: in this context, the overall similarity between the disorders is an important piece of knowledge which should not be distorted by premature differentiation. However, it is also necessary to be able to identify which of the possible disorders is in fact the relevant one, and for this purpose, differentiation is highly appropriate. We need to be sensitive not only to the possible biases introduced by the grid method, but also to the purpose and ends to which the knowledge is be put.

The overall conclusion produced by these results is that, even with a rating system being used which should have given the expert reasonable freedom in expressing his knowledge, most of the results produced by the automatic analysis need to be treated with considerable caution.

CONCLUSIONS

The main implication of these results is that the type of repertory grid used should be carefully matched to the types of knowledge in the domain. Moreover, the function and purpose for which the knowledge is to be used should be borne in mind. Experts in repertory grid theory and use have long been aware of some of these issues. The problem is that as the tools become more widely disseminated in knowledge engineering the danger arises that a naive application of the tools and technique will generate spurious results.

As we have indicated the kind of issues the knowledge engineer needs to be particularly careful about have to do with matching grid formats and assumptions to features of the domain of application.

If the knowledge consists entirely of continuous dimensions without discrete classes or absolute values, for example, then either a rating or a ranking system should be usable. If the dimensions involve absolute values, ratings are preferable. For dichotomous dimensions, a bipolar representation is indicated. Although use of ranking without tied values caused problems in this case study, it is a useful device for forcing choices in situations where the subjects may be unwilling to give extreme values, or for avoiding lop- sided grids (Bannister and Mair, 1968).

If the domain contains dimensions which cause problems for repertory grids, then complementary techniques should be used. Ironically, some of the techniques best suited to handling such problems are also now in the repertoire of KA tools.

One pair of such techniques consists of card sorts and item sorts. These are often viewed as a sort of poor person's repertory grid, but are more usefully viewed as complementary. Both techniques are ideally suited to handling non-continuous classes. Item sorts (in which the expert sorts actual items, or photos of items, rather than cards) have the added advantage that the elicitor does not need to know anything about the domain: the expert can simply be given a few specimens chosen literally at random, as an initial way of eliciting some domain constructs. For card sorts, the elicitor needs to know some domain terms, and there is always the faint risk of accidentally choosing terms from two mutually incompatible domain paradigms or terminology systems, which can lead to unpredictable results.

The laddered grid techniques (Hinkle, 1965) are also useful in a variety of ways: for example, they can be used to elicit a hierarchical framework from which suitable same-level elements can be extracted for use in the repertory grid. They are also good at handling categorical information. Laddering has been evaluated in KA by Burton, Shadbolt, Rugg, and Hedgecock, (1990) and is included in Aquinas (Boose, Shema and Bradshaw, 1989).

Developments in implemented repertory grids are ongoing, some of them prompted by precisely the sort of issues discussed earlier. One such example is Aquinas (Boose, Shema and Bradshaw, 1989), which allows users to mix cell value types within the same grid and which also supports other features such as laddering. Within the ESPRIT II ACKnowledge project (Shadbolt and Wielinga, 1990) there is an attempt to provide a set of synergistic and integrated KA tools. Moreover, the ambition is to engineer some of

the techniques to furnish *knowledgeable knowledge acquisition tools*. Such tools would be sensitive and informed by precisely the sorts of considerations discussed in this paper.

In the meantime we think it important to promote a more sophisticated and critical view of the tools and techniques that are on offer. As the repertory grid technique and associated tools become more widespread in knowledge engineering we must understand the implications of uncritical application. As we have seen in our small case study this might well produce what appears to be knowledge, and might even produce some interesting automatic analyses: but it could all too easily be a farrago of valid knowledge inextricably entwined with dubious dimensions and distorted cell values, producing misleading outputs.

Obviously, this study is only suggestive of the problems that can arise when using grids for knowledge engineering. There is a need to conduct larger scale evaluations of a number of grid implementations to refute or substantiate the worries expressed in this paper. In any event we believe that research is needed to understand how to take best advantage of these powerful techniques.

ACKNOWLEDGEMENTS

We would like to thank Dr. Brewer, Dept of Geology, University of Nottingham, for his time and patience as we sought to elicit his expertise in the area of mineralogy. Thanks are also due to Mike Burton and Colin Corbridge who commented on an earlier draft of this paper. This research was carried out, in part, whilst the authors were engaged in an ESPRIT funded project - P2576 ACKnowledge.

REFERENCES

Bannister, D. & Mair, J.M.M (1968) *The Evaluation of Personal Constructs*. Academic Press: London.

Boose, J. (1985) A knowledge acquisition program for expert systems based on personal construct psychology. *Int. J. Man-Machine Studies* 23:495–525.

Boose, J. (1986) Rapid acquisition and combination of knowledge from multiple experts in the same domain. *Future Generation Computing Systems* 1:191–216.

Boose, J. (1989) A survey of knowledge acquisition techniques and tools. *Journal of Knowledge Acquisition*, 1, 3–38.

Boose, H. & Bradshaw, J. (1988) Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems, in J. Boose & B. Gaines *Knowledge-Based Systems Vol 2*. Academic Press: London.

Boose, J., Shema, D. & Bradshaw, J. (1989) Recent progress in Aquinas: a knowledge acquisition workbench. *Journal of Knowledge Acquisition*, 1, 185–214.

Burton, A.M., Shadbolt, N.R., Rugg, G. & Hedgecock, A.P. (1990) The efficacy of knowledge elicitation techniques: a comparison across domains and levels of expertise. *Journal of Knowledge Acquisition*, 2, 167–178

Gaines, B. R. & Shaw, M. L. G. (1981) New directions in the analysis and interactive elicitation of personal construct systems, in M.L.G. Shaw (ed) *Recent advances in personal construct technology*. Academic Press: New York.

Gaines, B. R. & Shaw, M. L. G. (1986) Induction of inference rules for expert systems. *Fuzzy Sets and Systems*, 8 (3), 315–328.

Gaines, B. & Linster, M. (1990) Development of second generation knowledge acquisition systems. In B. Wielinga, J. Boose, B. Gaines, G. Schreiber & M. van Someren, (ed),Current Trends in Knowledge Acquisition, pages 143–160. Amsterdam: IOS Press.

Hinkle, D. (1965) *The change of personal construct fron the viewpoint of construct implications.* Unpublished Ph.D thesis: Ohio State University.

Murphy, G.L. & Wright, J.C. (1984) Changes in conceptual structure with expertise: Differences between experts and novices. *Journal of Experimental Psychology*, **10**, 144–155.

Reichgelt, H. & Shadbolt, N. (1991) ProtoKEW: A knowledge-based system for knowledge acquisition. In Sleeman, D. and Bernsen, O (eds) Research directions in cognitive science: artificial intelligence. Lawrence Erlbaum: London.

Shadbolt, N. & Burton, M. (1989) The empirical study of knowledge elicitation techniques. *SIGART Newsletter*, 108, April 1989, ACM Press.

Shadbolt, N. & Burton, M. (1990) Knowledge elicitation. In J. Wilson and N. Corlett, (eds), *Evaluation of Human Work: A Practical Ergonomics Methodology*, pages 321–346. Taylor and Francis.

Shadbolt, N. & Wielinga, B. (1990) Knowledge based knowledge acquisition: the next generation of support tools. In B. Wielinga, J. Boose, B. Gaines, G. Schreiber & M. van Someren, (ed), Current Trends in Knowledge Acquisition, pages 313–338. Amsterdam: IOS Press.

Shaw, M.L.G (1980) *On becoming a personal scientist: interactive computer elicitation of personal models of the world.* Academic Press: London.

Shaw, M.L.G. & Gaines, B.R. (1988) *A methodology for recognising concensus, correspondence, conflict and contrast in a knowledge acquisition system.* Paper presented at the Workshop on Knowledge Acquisition for Knowledge-Based Systems, Nov7–11, 1988, Banff.

Thomas, L.F. & Shaw, M.L.G (1976) *FOCUS: A manual for the FOCUS computer program*, Centre for the Study of Human Learning, Brunel University, Uxbridge.

Yorke, D.M. (1978) Repertory grids in educational research: some methodological considerations. *British Journal of Social and Clinical Psychology*, 9, 108–21.

Yorke, D.M. (1983) *The repertory grid: a critical appraisal.* unpublished Ph.D thesis, University of Nottingham.

Yorke, D.M. (1985) Adminstration, analysis and assumption: some aspects of validity, in N. Beail (ed) *Repertory grid technique and personal constructs: Applications in clinical and educational settings.* Croom Helom: London.

On the Acquisition and Representation of Procedural Knowledge

Tim Saito
Computer Sciences Corporation, 16511 Space Center Boulevard - M30, Houston, Texas 77058, U.S.A. (tsaito@nasamail.nasa.gov)

Chris Ortiz
Software Technology Branch/PT4, NASA/Johnson Space Center, Houston, Texas 77058, U.S.A. (ortiz@gothamcity.nasa.gov)

R. Bowen Loftin*
Department of Natural Sciences, University of Houston-Downtown, One Main Street, Room 813-N, Houston, Texas 77002, U.S.A., and NASA/Johnson Space Center (bloftin@nasamail.nasa.gov)

Historically knowledge acquisition has proven to be one of the greatest barriers to the development of intelligent systems. Current practice generally requires lengthy interactions between the expert whose knowledge is to be captured and the knowledge engineer whose responsibility is to acquire and represent the expert's knowledge in a useful form. Although much research has been devoted to the development of methodologies and computer software to aid in the capture and representation of some types of knowledge, little attention has been devoted to procedural knowledge. NASA personnel, on the other hand, frequently perform tasks that are primarily procedural in nature. In this paper we will review previous work in the field of knowledge acquisition and then focus on knowledge acquisition for procedural tasks with special attention devoted to the Navy's VISTA** tool. Finally, we will describe the design and development of a system for the acquisition and representation of procedural knowledge—TARGET (Task Analysis and Rule Generation Tool). TARGET is intended as a tool that permits experts to visually describe procedural tasks and as a common medium for knowledge refinement by the expert and knowledge engineer. The system is designed to represent the acquired knowledge in the form of production rules. Systems such as TARGET have the potential to profoundly reduce the time, difficulties, and costs of developing knowledge-based systems for the performance of procedural tasks.

INTRODUCTION

In order to set the stage for a description of "yet another" knowledge acquisition tool, a brief review of our evaluation of many existing knowledge acquisition (KA) tools is presented below. Following this review, a Navy-developed program for task analysis is discussed in some detail.

---

* Address all inquiries to R. B. Loftin, Mail Code PT4, NASA/Johnson Space Center, Houston, TX 77058, (713) 483-8070, bloftin@nasamail.nasa.gov.

** VISTA (Visual Interactive System for Task Analysis) was developed by Robert Ahlers, Naval Training Systems Center, Orlando, Florida.

The remainder of the paper is devoted to a description of our own work in creating a knowledge acquisition tool specifically for procedural tasks. In addition to its role in aiding experts and knowledge engineers in the process of knowledge acquisition, this tool also provides mechanisms to support knowledge refinement and its representation in the form of production rules.

Processes and software designed to aid knowledge acquisition can be characterized by the nature of their delivery and implementation methods and styles as well as their ability to extract knowledge. Various authoring tools have evolved to solve the problems associated with the creation of a specific expert system (Boose, 1989). Historically, most knowledge-acquisition-oriented tool designs were directed toward rating or categorizing problems or knowledge. To use such tools to capture specific knowledge, the developer distinguished between types of knowledge methods/approaches. Although sharing many of the same goals, the existing methodologies are numerous, ranging from frame modeling to case-based reasoning models to repertory-grid rating structures. The various knowledge types addressed by these systems—from semantic/taxonomic to declarative to procedural—affect the design and performance decisions of researchers and implementers (Gaines, 1988). Knowledge representations, including frames, objects, rules, and decision trees, are used to capture and execute expertise. At this point, most would agree that no one tool accommodates all of the cognitive styles needed to gather the information/knowledge necessary for the creation of an expert system in one contiguous process. It is clear that viable standards have yet to be fully established and accepted.

Procedural knowledge acquisition via task analysis is a reasonable candidate for graphical representation modes. Decomposing a complex set of steps that make up a specific mission or task requires cognitive visualization and the ability to formulate and reformulate the decomposition of those steps or actions. The specific heuristic procedures that most subject matter experts (SMEs) employ share certain levels of organization and recall (de Kleer, Doyle, Steele, Jr., and Sussman, 1985). The path in which a procedure evolves starts with specific agendas and goals. The last or final action of reaching or satisfying those actual goals would end the procedure. On the other hand, any actions that would restart a process (i.e., a loop) would occur before the goal-oriented or last action. Decisions may be made during a task that direct the expert along alternative paths that may or may not be taken in other performances of the same task. In cases where the processes offer one or more options to complete a task, the process diverges into as many paths as necessary to meet the optional requirements. Each path would then contain specific values for technique evaluation or other modes of feedback. These types of complexities lend themselves to representation in a visual form. Below, we explore two attempts to provide just such a visual metaphor for knowledge acquisition and representation.

PROCEDURAL ANALYSIS TOOL SEARCH AT NASA

As in other environments, getting and maintaining an SME's attention, time, commitment, help, and data sources at NASA is usually difficult at best. SMEs tend to differ in their communication abilities and styles, willingness to cooperate, availability, and degree of computer literacy, potentially affecting the overall success of the knowledge acquisition process (Littman, 1988). The strategy of providing the SME with a tool that can be used to document his mission(s) or task(s), on his own and within his schedule, would serve to resolve some of the difficulties associated with a knowledge engineer constantly "hovering over" an SME. However, the disadvantages of such a strategy may include the lack of positive reinforcement or external motivation (i.e., SMEs might put off documenting their task/mission unless periodically reminded or encouraged).

Other constraints affecting the type of tools delivered and used could be those associated with budget problems or deficits. Where exotic workstations might be required for more sophisticated KA tools, NASA may only have dated or under-powered PC hardware in certain areas of need. Distribution of KA tools to NASA personnel who have access to inadequate equipment to use in documenting their procedures could prove frustrating. SMEs tend to use their PCs or MacIntoshes

for spreadsheets, databases, or word processing where KA is not a daily issue in their operation. The need to deliver tools that do not require SMEs to alter their current work style and environment (hardware, software, operating systems, methods, etc.) is, therefore, critical.

As part of a technological assessment of current systems that lend themselves to KA, over 20 tools, varying in styles of implementation, were evaluated at NASA/Johnson Space Center. A goal of this activity was to identify tools that would:

1) Elicit or document procedural knowledge within an existing operation established by an SME or group of SMEs;
2) Run on current hardware configurations commonly-available throughout NASA with little or no modification; and
3) Offer an easy-to-use and robust environment (via a graphical user interface, intelligent front-end, etc.) for acquiring procedural knowledge.

As with other types of software tools and products, knowledge acquisition tools, either on the market or under development, seemed to come in many "flavors," "colors," and "shapes." Most of the KA tools that we examined could be characterized according to the following basic criteria:

1) Mode or style
   A) Front-end tools
   B) Embedded modules

2) Platform levels
   A) PC or MacIntosh
   B) Symbolic processing or workstation
   C) Mainframe compatible

3) Operating systems
   A) PC/Mac:
      • MS-DOS®
      • MacOS®
   B) UNIX compatibles
   C) Mainframe compatibles
      • VMS® (Virtual Memory System)
      • VM® (Virtual Memory)
      • MVS® (Multi-Virtual System)

4) Developer type
   A) Commercial
   B) Institutional

Some KA tools operate either as front-ends or embedded modules within existing expert systems and/or expert system development packages. Tools like Nextra, a commercial tool marketed by Neuron Data, serves as a front end to the Nexpert Object expert system. Aquinas, a rather powerful KA tool, also operates as a front-end tool in eliciting knowledge for expert systems development. Existing expert system development tools like ADS® (Aion Development System) by Aion Corporation perform their KA function in a built-in query and knowledge organizing module called an object or state editor. Smaller tools like Socrates™ (CIM Solutions) and VP-Expert™ (Paperback Software) also deliver similar modules for building knowledge bases. As the size of the hardware platform grows, so often does the complexity and sophistication of the knowledge acquisition tool or module that it supports.

Delivery platforms for the KA tools examined in our study ranged from PCs and Macintosh® computers to RISC and symbolic workstations to IBM® plug-compatible mainframe systems. Operating system environments included Microsoft DOS®, Macintosh® OS, UNIX®, and IBM's VM® (Virtual Memory) and MVS® (Multi-Virtual System). Some knowledge acquisition tool developers have released their tools under more than one operating system. Nextra™, for example, functions in multiple environments, including UNIX®, VM® (Virtual Machine - IBM), VMS® (Virtual Machine System from DEC®) and Macintosh® operating systems.

Such commercial PC-based tools as GURU® (mdbs, Inc.), VP-Expert™, and Socrates™ possessed modest levels of knowledge acquisition features generally oriented toward knowledge representation in the form of rules. On the other hand, mainframe compatible ADS®, Mercury KBE™ (Knowledge Base Enterprise from Artificial Intelligence Technologies) and Knowledge Shaper (Perceptrics, Inc.) all provided more elaborate knowledge acquisition functions and features (e.g., object-oriented representation and management) than the smaller systems. Tools under development such as KART (Knowledge Acquisition Reasoning Tool from IBM) and AKAT (Automated Knowledge Acquisition Tool from Harris Corp.) represent tools that operate within the more commonly-available workstation environments. However, other institutional knowledge acquisition tools such as Aquinas (Boeing) and KNACK (Carnegie-Mellon) provide sophisticated examples of the genre. DART (Design Alternative Reasoning Tool - Boeing ATC) is one spinoff tool whose design architecture was derived from its more renowned predecessor, Aquinas.

Identifying tools that provided a suitable environment for procedural knowledge capture proved difficult. Comparing tools of such diversity in styles and implementations complicated an already complex picture. However, one tool with a simple but effective interface and procedural metaphors was discovered. We were able to make use of it in spite of the absence of documentation. Having all the right components, report functions and features, this tool provided enough intuitiveness for a user to navigate and recover from any problems while using it. The graphically-oriented task analysis tool developed by Robert Ahlers of the Naval Training Systems Center (NTSC) in Orlando, Florida, showed the most promise for addressing procedural KA. VISTA (Visual Interactive System for Task Analysis) proved to be a tool an expert could use to easily define and document specified tasks in a "comfortable" and modifiable form.

VISTA: THE GRAPHICAL PREDECESSOR

The NTSC in Orlando, Florida, participating in a governmental tri-service project with the Air Force Human Resources Lab and the Army Research Institute, directed a project that produced a prototype knowledge acquisition tool called KA-1 (Knowledge Acquisition-1). KA-1 was first designed and implemented in Lisp within the Symbolics/Genera 6.1 environment. Recognizing the appropriateness of implementing their own knowledge acquisition strategies to acquire task or procedural knowledge, NTSC, after the end of the KA-1 project, started work on its first prototype (named AFEAT—Automated Front-End Analysis Tool). NTSC later redesigned and released an enhanced version, renaming the knowledge acquisition tool VISTA. Both of these applications have been developed on PC platforms using the Smalltalk/V 286 development package.

The NTSC designed VISTA to compose task lists and hierarchies from a graphical representation of a knowledge base. The VISTA tool utilizes a GUI to construct box-flow diagrams to capture the decomposition of tasks and procedures into their component parts in a largely top-down style. When acquiring procedural knowledge, it can be used with or without the intervention of a knowledge engineer. VISTA also lends itself to real-time verification and modification by a group of experts collectively viewing the graphical task representation.

VISTA maintains a fragile balance between ease of use and design complexity/intricacy. Although it does not possess the "bells and whistles" of more sophisticated systems like Aquinas and

Protege, VISTA provides enough knowledge modeling (procedural/declarative) support to allow the SME or knowledge engineer to build a moderately elaborate knowledge base without sacrificing the attractiveness of its user interface.

VISTA provides a "windows-icon-mouse-pointing" (christened, WIMP) interface environment based on a grid-marked Work Area in which the user builds task networks. The WIMP approach, a three-button mouse architecture, facilitates the rapid selection and execution of system functions to minimize user keystrokes. The Work Area is lined on three sides with icon and menu selectable functions:

1) System Command Menu Bar (top),
2) Function/Graphics Icons (left side), and
3) Message/Explanation Bar (bottom).

Although VISTA provides no bona fide compilation facilities to check the knowledge base for completeness or accuracy, it does utilize some of the Smalltalk inspection and reporting features to help the user confirm the knowledge input into the system. VISTA provides a windowed environment through which decomposition can be organized and recorded. Ultimately the user, knowledge engineer or SME, is responsible for the overall quality checking of the knowledge base before its representation in or transfer to other applications. VISTA's report facilities offer some assistance in this quality checking process. Reports can be generated to provide moderately high-level feedback to the knowledge engineer and SME. VISTA produces the following reports:

- Hierarchical Statistics: counts nodes and subnets at each level;
- Task hierarchy: keeps a sequential/hierarchical account of tasks;
- Input grammar: maintains various types of component titles in categorized form;
- Notecards: keeps notes on conditions, states or other user-supplied details; and
- Highlighted tasks: accounts for subnet levels.

VISTA supports the identification and conceptualization phases of knowledge acquisition with its network approach to knowledge representation. Duties, tasks/subtasks, or steps/substeps within a process can be defined, documented, and structured to reflect these relationships to other duties, tasks/subtasks, or steps/substeps.

Given its developmental state, VISTA provides a reasonably comprehensive mechanism for generating simple representations at the very first knowledge acquisition session. The next sessions may be used to embellish what has already been elicited or to create new or modified versions of the knowledge base. The VISTA knowledge acquisition interface gives the user the freedom to generate as complex a hierarchy of knowledge as necessary. However, the disadvantage to such freedom is the ability to create a completely abstract knowledge base with relatively few standards for input. Some guiding controls from the VISTA interface could provide structure to the knowledge acquisition process and greatly enhance the ability of the user to create a "useful" knowledge base.

Although VISTA has a grammar component within its facilities, its ability to correlate domain terms and/or concepts is limited. The open-endedness of the tool design allows key domain definitions to be specifically addressed within a notecard-like management facility and/or defined in a box as a subpart within a process. VISTA does not offer a true lexical facility or natural language interface to accommodate concept definitions.

Although VISTA had obvious strengths with its GUI-oriented features, there were also some significant limitations, not only in design but also in logistics, that prompted our plans to build a tool that contained some of the "look and feel" of VISTA while addressing VISTA's shortcomings and including the ablility to represent the knowledge base in the form of production rules and

procedural code. At the time of our evaluation, VISTA's task hierarchy report generator was not a high development priority, and there were problems with its accuracy. For example, "go-to" and decision box representations were not included in the task hierarchy output report although they were represented in the task flow diagrams. The Smalltalk V programming language and environment were not conducive to porting applications between PCs, Macs, workstations, and mainframe systems. In addition, VISTA's interface was based on a three-button mouse interaction, making it difficult to implement on a MacIntosh. The developers designed the tool for the Navy environment with emphasis on mission or task analysis. The use we envisioned for a KA tool within the NASA environment was significantly different from that which led to the Navy's creation of VISTA.

NASA ENVIRONMENT AND NEEDS

The National Aeronautics and Space Administration commits large funding and manpower resources to training new and existing personnel. New recruits are trained to carry out tasks for which they were hired. Existing staff must be trained or retrained to upgrade/update abilities to perform current and new tasks. Significant numbers of training methodologies are utilized involving training manuals, formal classes, instructional computer programs, simulations and on-the-job training. Simulation-based training is usually the most effective mechanism for the more complex tasks requiring substantial autonomy on the part of the task performer. However, this training style is also the most expensive and may be impractical where trainees significantly outnumber experienced staff and opportunities to be properly trained are scarce.

The effort of educating and training NASA astronauts, flight controllers, and other ground support personnel has generally required extensive on-the-job experience in order for individuals to acquire the knowledge and skills necessary for acceptable performance and/or certification. Current flight schedules, combined with the loss of experienced personnel to retirement or transfer, have significantly reduced the ability of traditional training techniques to produce an adequate number of trained personnel (Loftin, Wang, Baffes, and Hua, 1988). As NASA prepares for acceleration in Space Shuttle and Space Station Freedom missions, the need to minimize training related risks has become more imperative. Employee turnover or program cuts can also significantly impact mission operations and underscore the need to preserve corporate memory.

Recently, it has been shown that workstation-based, intelligent computer-aided training systems (ICATs) can deliver effective, intensive, and personalized training to large numbers of trainees, independent of integrated simulations (Loftin, Wang, Baffes, and Hua, 1989). ICATs have proven their efficacy in augmenting traditional simulation-based training. Such systems can noticeably reduce the amount of training time needed to achieve acceptable levels of performance. After over five years of experience in building such systems, the developers have concluded that the greatest barrier to the large-scale, efficient production of ICAT systems is the extent and difficulty of the knowledge acquisition process. For some time an effort has been underway to create a software tool to aid in the capture of mission-support procedural knowledge, both to preserve the existing corporate knowledge base and to assist in the development of decision-support expert systems and ICAT systems.

In the NASA/Johnson Space Center environment, CLIPS (C-Language Integrated Production System) is widely used as an expert system development and delivery vehicle. Within the ICAT metaphor the overall procedure or task is decomposed into sets of tasks/subtasks that are termed actions. For most effective use, actions are expressed, within reason, at the lowest possible level. At any point in an ICAT training session, the expert expects the trainee to perform some action. Each action, as defined by the expert, is represented as a CLIPS fact in the following pattern:

(message-sender-to-receiver <step number> <action type> <argument> <argument> ...)

An action itself comprises at least two <argument> fields that define one single action decomposed into a hierarchical structure of two or more subactions of the form (<action> <argument>). Each <argument> may itself be an <action> at the next lower level. For example, (arg1 arg2 ... argn) can be decomposed into at least two levels: (arg1 arg2) and (arg2 . . . argn). The first pair, (arg1 arg2), is an (<action> <argument>) pair at the top level where the action arg1 has one argument, arg2. In turn, (arg2 ... argn) is another (<action> <argument>) pair at the second level where arg2 has one or more arguments, depending on the value of n. The structure for each action may be different and the number of arguments that belong with each action is variable. The expert is free to decompose the actions and arguments into hierarchies that fit his or her specific domain.

The remainder of this paper details the design and implementation of a knowledge acquisition system tailored to the acquisition and representation of procedural knowledge associated with the performance of complex tasks. The primary goal of this effort has been the production of a system with an easy-to-learn and "comfortable" user interface that provides powerful mechanisms for the visual expression of procedural knowledge. The ultimate goal of this work is the expression of acquired knowledge in the form of production rules to facilitate the use of the acquired knowledge in expert systems for mission support and training.

THE TARGET APPROACH

TARGET and Design Strategy

Attempting to strike that delicate balance between non-programmer usability, design sophistication, and hardware universality, the Task Analysis Rule GEneration Tool (TARGET) is designed to provide a knowledge acquisition environment for users of commonly-available computer systems (IBM® PCs and Apple© Macintoshes®). The forte of TARGET is the gathering of task or procedural knowledge to be expressed and analyzed graphically as well as contextually. TARGET provides users the ability to graphically decompose a task or procedure using a box-flow presentation/manipulation style within a windowed environment.

TARGET is designed to let the SMEs start documenting their job or task with minimal training in its use and no absolute need for knowledge engineer intervention. If the SME is unable to find time to work on the knowledge acquisition process alone, TARGET does allow the knowledge engineer and SME to work together in iterative sessions. It is tailored to accommodate a wide range of users, from the novice to the expert. Users can develop a discrete representation of tasks and subtasks within their domains (Payne and Green, 1986). The system then manages the information entered and represents the knowledge in a "top-down" reporting format that can then be used for rule induction and generation.

TARGET supports three action types: required, optional and flexible (as defined in the ICAT architecture). In order to support the development of intelligent computer-aided training (ICAT) systems, TARGET implements its rule representations using the rule types and structures originally developed for ICAT systems. TARGET is designed to deal with a series of tasks/subtasks that are performed procedurally or in steps. Steps are defined as the progression from one required action to the next. Steps are represented by numerals and their values increase with the progression of the task.

Task-Action Concepts (Building Blocks)

Like VISTA, TARGET employs a free-form flow charting concept. SMEs or knowledge engineers can explain procedural processes by the use of various icons arranged in the TARGET work area. The tasks can then be linked together using directed arcs to show procedural flow. The task icons are separated into four major categories.

The first category consists of the actions that are required to complete a process (Figure 1). The required actions are denoted by using a blue (or white, for monochrome monitors) rectangular box on the screen. On the other hand, optional tasks are represented using grey rectangles. The use of color for representing various forms of tasks has been reduced to just two, grey for optional tasks and blue (white) for all others. The shape of the task box is the most important key to determining its function. By using shapes and not colors, TARGET, although designed for color systems, also works reasonably well on monochrome systems.

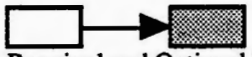


Fig. 1 Required and Optional Tasks

Hexagonal-shaped structures are used to show the points in a process where decisions are needed. The connections leaving the decision structures are labeled to show what action is to be taken. For example, a decision structure may ask if a process has been completed. This decision may be linked to two other tasks (Figure 2). The first arc leaving the decision is labeled "YES" and the other is labeled "NO". The arc labels represent the only possible answers allowed by the decision structure.

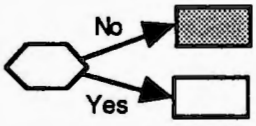


Fig. 2 Decision Task

The fourth basic structure that TARGET provides is a control structure. Control structures are used as a "go-to" or looping mechanism. Controls can only jump to other tasks that are on the current layer (see discussion of layers below) and may not jump directly to other controls. Control tasks are denoted by using an ellipse to distinguish them from other tasks (Figure 3).

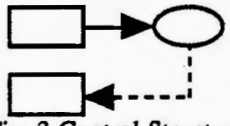


Fig. 3 Control Structure

In addition to the two-dimensional flow, TARGET provides users with the ability to decompose required and optional tasks into a task hierarchy. Task decomposition is used to make complex tasks easier to understand and complicated layers smaller. Tasks that are decomposed are characterized by displaying a "shadow" that represents the "child" tasks associated with them. The child tasks are not seen at the "parent" layer but can be found when traversing down the task hierarchy into lower levels (Figure 4).

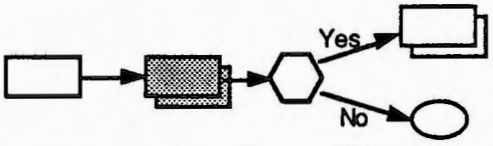


Fig. 4 Task Chain Showing Child Tasks

Navigating through a task hierarchy is done simply by placing the cursor on a leveled task and double-clicking. The screen will clear and be rebuilt showing the selected child tasks (Figure 5). Moving up in the task hierarchy is just as simple, it is done by double-clicking on an area where there is no task. The screen will clear and tasks from the level above will be displayed.

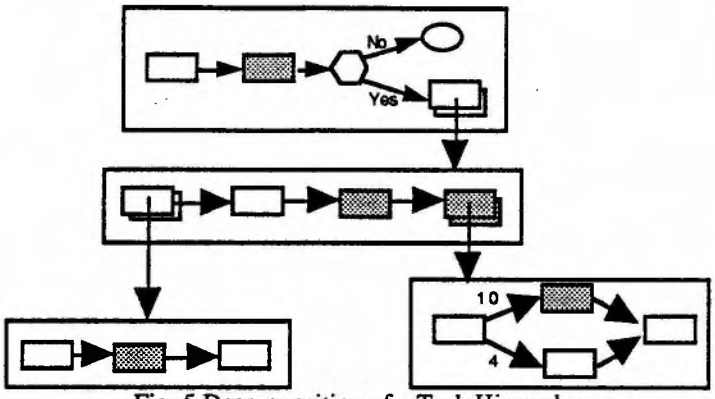


Fig. 5 Decomposition of a Task Hierarchy

Significant Interface Features

To facilitate its implementation on many hardware platforms, TARGET was developed around the one-button mouse concept. Using a single mouse button was determined to be the best way of porting the TARGET program to other platforms such as a Macintosh™ computer. Double-clicking the mouse button will move up or down a task hierarchy and single-clicking will activate all other functions. Since single-clicking on a one-button mouse is limiting, TARGET, like VISTA, was designed to be a mouse-driven program.

Users may select a desired function from the main menu, toolbox or keyboard (Figure 6). The cursor will change its shape to reflect the operation that is to be performed. Located on the left-hand side of the screen, the toolbox is one of the most important user interface functions provided by TARGET. Using the toolbox, one can directly manipulate task structures on the current screen by editing, creating, deleting, moving, linking, and un-linking tasks. Help and a display of the task hierarchy may also be accessed through this toolbox.

Menu selections provide an interface for file input/output, for task manipulation, to set user preferences for how TARGET displays the task flow, to control output, and to obtain help. File I/O for reading and writing TARGET files (.TGT) to and from disk storage is accomplished through the FILE menu. The EDIT menu helps to find tasks as well as permit the moving of tasks up and down the task hierarchy. The VIEW menu is especially helpful. Using VIEW, one can display tasks in a horizontal or vertical orientation. Optional tasks and the background grid can be hidden or displayed. The entire layout may be redrawn using automatic task placement. OUTPUT provides a mechanism to generate reports or simple rule structures on the screen, printer, or in ASCII format on disks. HELP provides information regarding TARGET questions as well as an explanation of the help facility itself.

TARGET provides the ability to enlarge or shrink the perspective for a better view of a task network. The ZOOM facility can be used to display a task network at 100% as its default size. However, it also displays 50%, 25% and 12% views of a network. In addition, all functionality of the tool, editing , moving, deletion, etc., would still be preserved.

The Check Network command under the Help menu allows developer to check the current network configuration for any task flow errors. If an error occurs, TARGET will transfer the user to the level where the violation was first detected. The user will select the Help menu to invoke the Check Network command. It will point to the violating task where the error occurred.

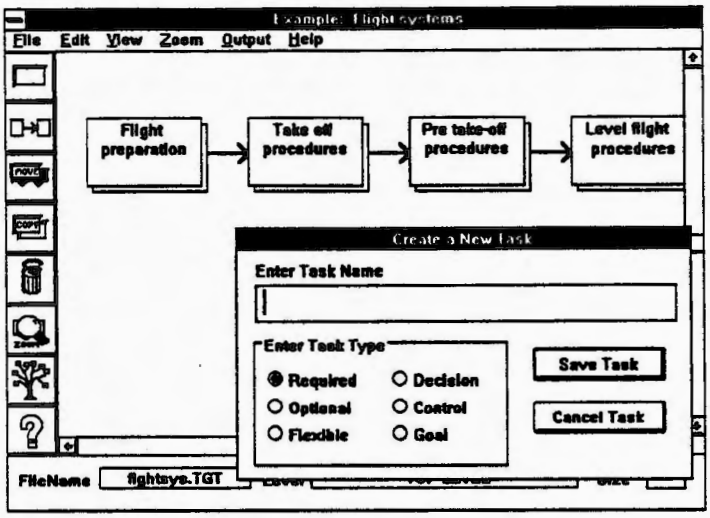


Fig. 6 TARGET User Interface

Task Documentation within a Process-flow Diagram

Before computer generated flowcharts or process modelers, most task analysis projects were performed via paper and pencil. The tedium and overhead requirements of recording and drawing box-flow representations of a complex task often created bottlenecks. With faster computer hardware and more sophisticated GUI based drawing tools, it has become feasible to decompose complex levels of task abstraction with more efficiency and ease (Bylander and Chandrasekaran, 1987). As part of this evolution, TARGET's visual programming design employs directed graph strategies driving its box-flow diagrams and entity-relationships within its task hierarchy. Boxes represent single entities or objects being linked to other boxes or entities by TARGET links.

TARGET provides features for organizing selected missions or tasks into distinct hierarchical levels. Individual steps can be laid out and studied interactively by the SME and/or knowledge expert. Immediate graphical feedback for analysis is possible. The user need not be concerned with "keeping score" on these issues, TARGET, through its graphical maintenance facility, will manage the top-down representation and coordination of tasks. It will also accommodate the text editing process as well as maintain box/task step order. Each box inherits the procedural value or discrete action that acts as a pre-condition to the next box or step. For example, a single TARGET box could simply contain the discrete step:

Action 1) Aim ballistic missile to strategic launch point

The action that a single box possesses would provide implied information to the next discrete step as:

Action 2) Fire ballistic missile at strategic launch point

where:

Pseudorule 1)

IF (implied)	Start
	=>
THEN	Aim ballistic missile to strategic launch point.

The action of aiming becomes the implied condition of the next rule (subsequent box):

Pseudorule 2)

IF (implied)	missile aimed at strategic launch point
	=>
THEN	Fire ballistic missile at strategic launch point.

Task hierarchies require large amounts of layout space while visual analysis is performed (Wilson, 1989). The basic human factors rule-of-thumb recommends the "7 plus-or-minus 2" boxes/tasks per level for creating hierarchical box diagrams. When documenting complex tasks, however, the user should not be limited (especially in his initial effort) to a specific number of boxes (Wilson and MacLean, 1985). Once accustomed to the approach TARGET takes in decomposing task flows, the domain expert can establish multiple mission/task levels hierarchically using his or her own classification or categorization strategies to express specific procedures. TARGET provides a traversing mechanism with which the user can create, maintain, and check task subnetworks with minimal keystrokes. Its strength lies in its ability to keep track of the task networks being developed. Therefore, a user could generate any number of boxes within a task level and convert an appropriate set of those steps into a subnet.

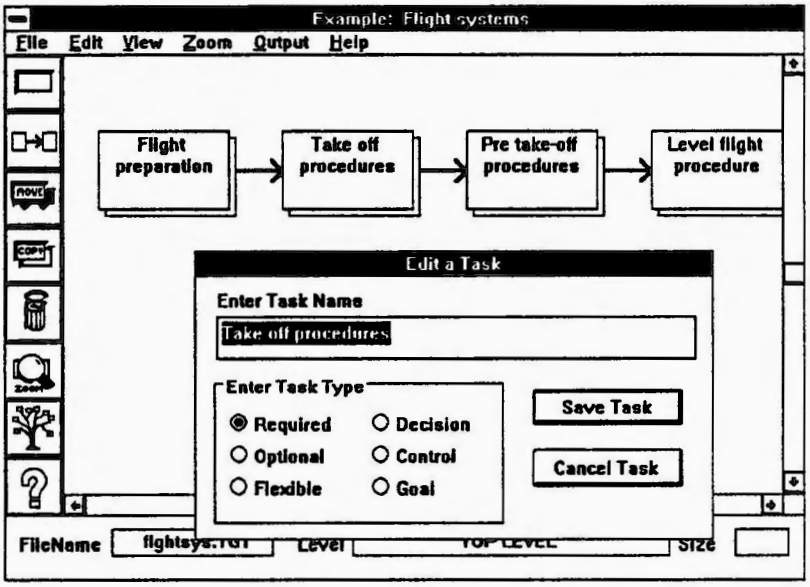


Fig. 7 Editing a Task in TARGET

TARGET offers two styles of box documentation formats. First, the user can describe singular discrete steps in a free form style with no input contstraints. For task analysts, this format would suffice in documenting procedures and generating top-down task outline reports. The "Enter Task Name" function provides a mechanism for the free-form input (Figure 7). The system does not check or verify content in this mode. However, within the TARGET environment, addressing another person would be the most effective way to phrase a task. Communication with that someone should be done in first person with an implied "you". For example, "You should":

Turn light on	Flip switch up
Report emergency to Commander	Allocate extended memory to DOS
Shut all systems down	Fire missile

Second, TARGET provides a template that breaks down the task steps into sequential dependencies to facilitate representation in CLIPS rules or facts. To avoid the need to support a detailed natural language (NL) parsing of the free-form task descriptions, a compromise to limit input to only significant command oriented expressions was implemented (Figure 8).

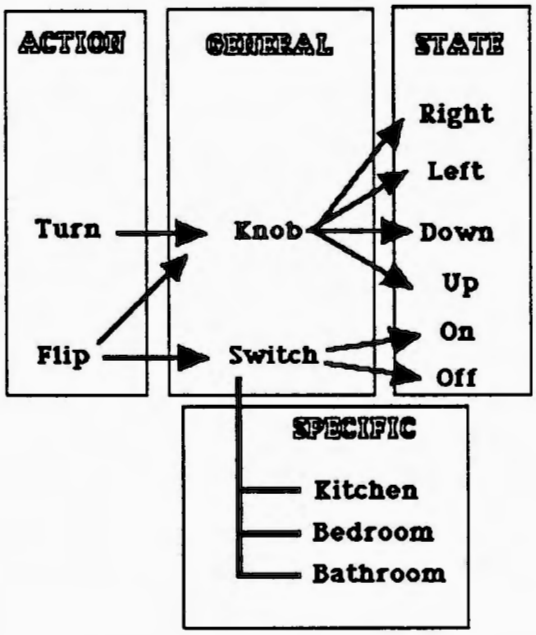


Fig. 8. Semantic architecture for procedural input

Task discourse structure would provides an economical means of expressing the basic task action in so many words (Allen, 1987). In using an NL derived discourse structure, the user, whether knowledge engineer or SME, is asked to input the following types of information (Figure 9) within the box:

1) Action verb: A simple present tense verb, in first person, that describes a discrete action that a person would perform as a component part of a process.

2) Object of an action: The focus of the particular action that the person would perform. For example in the task description, "Turn light on", the "light" would be the object acted upon.

   A) General object: In "turn light on", the "light" would be a general object.
   B) Specific object: In "turn kitchen light on", the "kitchen" designation would be an instance of the "light" object.

3) State of the object: A description of the object after it has been acted upon. For example, in "turn kitchen light on", the "on" state describes the desired state of that object.

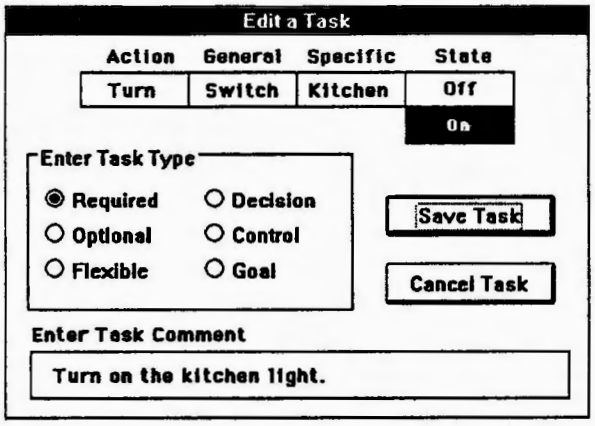


Fig. 9. Input template for task documentation

As a result, each box encapsulates the essential information, needed to characterize a specific task or action.

The user has slots in which to input additional knowledge elements:

1) Time limits or constraints (seconds/minutes/hours/days/months/years)
2) Criticality or salience factors
3) Rationale (to be encoded in CLIPS as comments)
4) Desired results (A description of the expected result to be included within the hypothesis in the next task)

Rule Control Structure Design

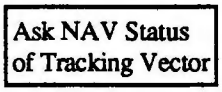
For rule propagation, TARGET implements the ICAT control structure guidelines where rules employ two parts, the data and the task process. In general, facts are data equivalents. The process or task step is manifested in the form of a rule. Facts are used to specify the current context or environment and to represent the "actions" taken by a user or an expert system. All fact combinations must be matched against rule patterns to determine which rules may fire. Then, one or more rules are used to generate potentially executable actions and update the fact base. This process is repeated until no rules can fire. More specifically, the ICAT control structure revolves around two significant points.

1) Message passing protocols are used by independent rule sets for communication
2) Tasks are procedural/step-by-step in structure.

Required actions are necessary task(s) and/or subtask(s) performed at specific points in a mission or task. These actions are then asserted to the factlist by the expert module. The following pattern reflects the expected action at a given point in time:

(message-sender-to-receiver <step number> require <argument> <argument> ...)

For example, Figure 10 shows the TARGET visual representation of the action: "Ask the Navigation Officer to give the status (good or bad) of the current state vector (describing a specific orbiting vehicle) obtained from ground- or space-based tracking stations." Following this representation is the CLIPS fact produced to express the same knowledge.



(message-sender-to-receiver 150 require request nav nav-status)

Fig. 10 Required Task

The ICAT architecture prohibits the performance of further actions if the current action cannot be achieved. In cases where more than one alternative precedes a required action, the alternatives may not all be equivalent. Although they accomplish the same goal, one method may be more suitable than another in a given context. TARGET distinguishes between these alternatives by labeling the "less desirable" as an "other" required action, as opposed to the previous "require" action. In the ICAT architecture, only one "require" action exists at any point with the rest, if any, being "other" actions. When more than one alternative exists for a required action (and none are more suitable than the others), all alternatives are labeled as "other" actions. Other actions are represented by TARGET as:

(message-sender-to-receiver <step number> other <argument> <argument> ...)

Optional actions are those recommended by the SME as good actions or technique preferences in a given context. However, the trainee is not prohibited from advancing in the training session if the action is disregarded. Optional actions should not be confused with situations where more than one required action exists at a given time. Optional actions are represented by the following pattern:

(message-sender-to-receiver <step number> optional <argument> <argument> ...)

Figure 11 shows the TARGET visual representation of an optional task.



Fig. 11 Optional Task

Since optional actions do not halt the training session within the ICAT architecture, even if they are ignored by the trainee, <step number> could be any future step beginning from the current context. For example, if the current step is 20 and the expert asserts (message-sender-to-receiver 50 optional ...), that particular optional action remains valid from step 20 until step 50. Validity checks for certain values can be displayed in the user interface. For example, a fact of the form

(message-sender-to-receiver 260 optional check-display vector-comp MET)

indicates that the expert recommends that the Mission Elapsed Time field within the Vector Comparison Table display be checked at any time from the current step through step number 260.

TARGET will internally manage flexible actions, i.e., actions that must be completed by predefined times (similar to a required action but operable over a span of time or steps). Flexible actions operate identically to optional actions inside a specified range of steps. If the deadline arrives and the flexible action still has not been completed, it will be changed into a required action. Flexible actions take the following pattern:

(message-sender-to-receiver <step number> flexible <argument> <argument> ...)

Report Generation with TARGET

TARGET provides several reporting mechanisms. The first report that the average user will experience is the graphical representation of tasks in a task hierarchy. Figure 12 represents a sample graphical layer that would be produced by a user. Task hierarchies in a textual form can also be generated from TARGET to provide another look at the procedural process being described (Figure 13). Files that store all the information TARGET needs to build task networks are called .TGT files (Figure 14). .TGT files are closely related to the .DMN files created by VISTA. The file similarity is intended to expedite the conversion from VISTA files into TARGET representations. The last figure (Figure 15) in this section shows CLIPS rules, written in final form, produced by TARGET.

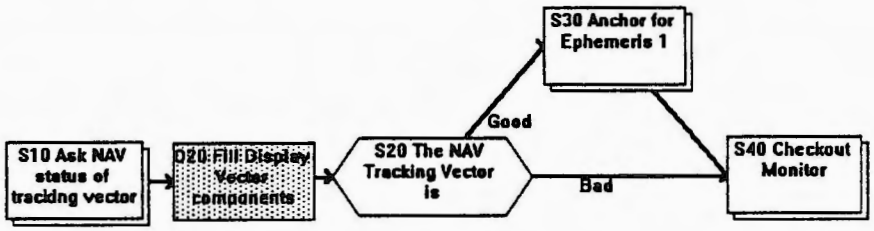


Fig. 12. Sample Task Chain Described Graphically in TARGET

1.0 Request NAV tracking vector status
- 1.1 (f) "toggle-offset-sign"
- 1.2 (f) "enter orbiter PKM offset in Worksheet"
- 1.3 (f) "enter deploy separation rate in Worksheet"
- 1.4 (f) "enter deploy spin axis declination in Worksheet"
- 1.5 (f) "enter PKM offset in Deploy Worksheet"
- 1.6 (f) "enter deploy relative right Ascension in Worksheet"
- 1.7 (f) "enter Z component of deploy delta V in Worksheet"
- 1.8 (f) "inform SDP DYN of payload data"
- 1.9 (f) "enter target PKM offset in Worksheet"

2.0 (o) check Vector Comparison Table

3.0 The NAV Tracking Vector is (Good)
- 4.0 Anchor ephemeris for shuttle
  - 4.1 (o) inspect new ephemerides
  - 4.2 (o) request Vector Comparison Table
  - 4.3 (o) request Trajectory Digitals display for ascending node
  - 4.4 (o) request Trajectory Digitals for descending node

3.0 The NAV Tracking Vector is (Bad)

5.0 request Checkout Monitor w/o Traj Digitals

Fig. 13. Task Hierarchy Description Generated from TARGET

REQUIRED
S10 Request NAV tracking vector status

2	<Task Key>
273,139	<X,Y Pos>
1	<Parent>
11,10,9,8,7,6,5,4,3	<Children>
13	<Connected_To>

```
OPTIONAL
O20 Fill Display Vector components
13                              <Task Key>
402,140                         <X,Y Pos>
1                               <Parent>
14                              <Connected_To>
DECISION
S20 The NAV Vector is Good
14                              <Task Key>
545,136                         <X,Y Pos>
1                               <Parent>
Good
18                              <Connected_To>
Bad
12                              <Connected_To>
REQUIRED
S30 anchor ephemeris for shuttle
18                              <Task Key>
691,55                          <X,Y Pos>
1                               <Parent>
17,15                           <Children>
12                              <Connected_To>
```

Fig. 12 .TGT File Derived from a VISTA .DMN File

```
(defrule s10-get-nav-info "Request NAV tracking vector status"
        (step ?s&10)
        (checkpoint expert)
=>
        (assert (message-sender-to-receiver ?s require req-nav get-nav-status)
                (next-step 20))
)

(defrule o20-display-vector-comp "Fill Display Vector components"
        (step ?s&20)
        (checkpoint expert)
 =>
        (assert
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_CUR_TIME)
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_CUR_DESC)
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_CUR_A)
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_3RD_TIME)
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_3RD_DESC)
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_3RD_A)
        (message-sender-to-receiver ?s optional check-display dis-vector-comp VC_3RD_V))
)

(defrule s20-good-nav "The NAV Vector is Good"
        (step ?s&20)
        (environment 0 nav-tracking good)
        (checkpoint expert)
=>
        (assert (message-sender-to-receiver ?s require req-nav put-nav-in-slot v39))
```

```
        (assert (next-step 30))
)

(defrule s20-bad-nav "The NAV Vector is Bad"
        ?f1 <- (step ?s&20)
        (environment 0 nav-tracking no-good)
        ?f2 <- (last-step ?)
        (checkpoint expert)
=>
        (retract ?f1 ?f2)
        (assert (step 40)
                (last-step ?s))
)
```

Fig. 13. TARGET Rule-Generated File

CONCLUSION

TARGET will have the capability to significantly impact the development of ICAT systems as well as the development of other intelligent systems. For any procedural knowledge acquisition task, it can enhance the ability of the expert to visualize and organize a task or process. We believe that procedural visualization of this type will become more popular as more tools with organizational diagnosis capabilities evolve (Akscyn, McCracken, and Yoder, 1988).

As computer hardware power evolves, more latitude in presentation methods will be available. Visual conception and communication of abstract information will become more common. The strategic fusion of graphical display (bit-map, meta-graphic, etc.) and graphical input device (mouse, light-pen, trackball, etc.) technologies will facilitate visual as well as textual representation of knowledge (Messinger, Rowe, and Henry, 1991). Drawing tools already allow the user to produce and manipulate complex graphics. The role of these tools can also combine with organizational algorithms to create more intelligent diagrams, flow charts and interactive decision trees. With users becoming more adept at employing systems with pictorial modeling capabilities, the mode of procedural KA will also benefit from such advances.

As knowledge acquisition evolves as a discipline within artificial intelligence, more tools to assist in the knowledge acquisition process will also become available in useful forms. TARGET, and tools like it, will be employed within their own "niche" and will also be integrated with other methodologies in the future. Although TARGET currently models the sequence within the task hierarchy structure for rule induction, we will dedicate additional efforts to encapsulating additional knowledge into the steps within a network. In particular, we intend to address issues such as gathering artifact data, selected action rationale, and interactive verification and validation of rules.

REFERENCES

Akscyn, R. M., McCracken, D. L. & Yoder, E. A. (1988). "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations", *Communications of the ACM*, July, 31 (7), 820-834.

Allen, James, (1987), Natural Language Understanding, Menlo Park, CA: The Benjamin/Cummings Publishing Company, Inc.

Boose, J. H. (1989). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, March, 1 (1), 3-37.

Bylander, T. & Chandrasekaran, B. (1987). Generic tasks for knowledge-based reasoning: the 'right' level of abstraction for knowledge acquisition, *International Journal of Man-Machine Studies*, 26, 231-243.

de Kleer, J., Doyle, J., Steele, G. L., Jr. & Sussman, G. J. (1985). AMORD: Explicit Control of Reasoning. in *Readings in Knowledge Representation*, Brachman, R. J. & Levesque, H. J., Eds., Los Altos, CA: Morgan-Kaufmann Publishers, Inc., 345-355.

Gaines, B. R. (1988). An overview of knowledge-acquisition and transfer. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R & Boose, J. H., Eds., *Knowledge-Based Systems, Vol.1*, New York: Academic Press, 3-22.

Littman, D. C. (1988). Modelling human expertise in knowledge engineering: some preliminary observations. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R. & Boose, J. H., Eds., *Knowledge-Based Systems*, Vol.1, New York: Academic Press, 93-104.

Loftin, R. B., Wang, L., Baffes, L. & Hua, G. (1988). An Intelligent Training System for Space Shuttle Flight Controllers. *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence*, held May 24, 1988, at NASA/Goddard Space Flight Center, Greenbelt, Md, 3-10.

Loftin, R. B., Wang, L., Baffes, P. & Hua, L. (1989). An Intelligent System for Training Space Shuttle Flight Controllers in Satellite Deployment Procedures. *Machine-Mediated Learning*, 3, 43-47.

Messinger, E. B., Rowe, L. A. & Henry, R. R. (1991). A divide-and-conquer algorithm for the layout of large directed graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (1), 1-11.

Payne, S., & Green, T. (1986). Task-action grammars: a model of the mental representation of task languages. *Human-Computer Interaction*, 2, 93-133.

Wilson, M. (1989). Task models for knowledge elicitation. in *Knowledge Elicitation: Principles, Techniques and Applications*, Diaper, D., Ed., New York: Ellis Horwood, 197-219.

Wilson, M. D., Barnard, P. & MacLean, A. (1985). Analysing the learning of command sequences in a menu system. Johnson, P., & Cook, S., Eds., *People and Computers: Designing the Interface*, Cambridge: Cambridge University Press, 89-102.

Using Integrated Knowledge Acquisition to Prepare Sophisticated Expert Plans for Their Re-Use in Novel Situations*

Franz Schmalhofer, Ralph Bergmann, Otto Kühn, Gabriele Schmidt

German Research Center for Artificial Intelligence
University Bldg 57
Erwin-Schrödinger Str.
D-6750 Kaiserslautern
Germany

e-mail: schmalho@informatik.uni-kl.de

Abstract: Plans which were constructed by human experts and have been repeatedly executed to the complete satisfaction of some customer in a complex real world domain contain very valuable planning knowledge. In order to make this compiled knowledge re-usable for novel situations, a specific integrated knowledge acquisition method has been developed: First, a domain theory is established from documentation materials or texts, which is then used as the foundation for explaining how the plan achieves the planning goal. Secondly, hierarchically structured problem class definitions are obtained from the practitioners' highlevel problem conceptualizations. The descriptions of these problem classes also provide operationality criteria for the various levels in the hierarchy. A skeletal plan is then constructed for each problem class with an explanation-based learning procedure. These skeletal plans consist of a sequence of general plan elements, so that each plan element can be independently refined. The skeletal plan thus accounts for the interactions between the various concrete operations of the plan at a general level. The complexity of the planning problem is thereby factored in a domain-specific way and the compiled knowledge of sophisticated expert plans can be re-used in novel situations.

1. MOTIVATION

Like other synthetic tasks, planning problems are inherently intractable (Georgeff, 1987). In a complex real world domain such as production planning in mechanical engineering, expert systems can consequently not be based on planning from first

---

* This research was funded by "Bundesministerium für Forschung und Technologie" under grant ITW 8902 C4 and in part by grant Schm 648/1 from "Deutsche Forschungsgemeinschaft". This paper was presented at the German Workshop for Artificial Intelligence, Sept. 16-20, Bonn, Germany.

principles (Koehler, 1991). It is also not surprising that in at least 80 percent of all mechanical engineering planning tasks, even human planners re-use old plans by adapting them to the new planning problem (Spur, 1979; Thoben & Schmalhofer, 1990).

Expert plans have not only been developed with much effort, but were also carefully tested and have proven their sophistication during numerous successful executions in the real world. Preparing such human planning solutions for their re-use in novel situations can provide an important basis for the development of a successful planning system.

This paper describes a general procedure by which concrete human expert plans can be generalized into skeletal plans (Friedland & Iwasaki, 1985). A skeletal plan provides a partitioning of the enormous search space of the complete planning problem into a number of subproblems with small search spaces. The skeletal plans constructed by this procedure are indexed by the various application conditions so that they can be re-used in novel situations.

Explanation-based learning (Mitchell, Keller, & Kedar-Cabelli, 1986) is applied to find an appropriate generalisation of a concrete case consisting of the description of a manufacturing problem and its solution. It is embedded into an integrated knowledge acquisition method (Schmalhofer, Kuehn, & Schmidt 1991) which provides the domain theory and allows the specification of domain-adequate operationality criteria for the construction of skeletal plans.

We will first outline the integrated knowledge acquisition framework, which is based on a quite general model of expertise. The general model describes the overall structure of the future expert system. We will then describe the skeletal plan construction procedure and its implementation in some detail. The application of the procedure to the production planning of a rotational part will be described and the results will be discussed.

2. INTEGRATED KNOWLEDGE ACQUISITION FOR PLAN RE-USE

The problem of production planning in mechanical engineering consists of finding an adequate production plan for a given workpiece which is to be manufactured in some factory. For the manufacturing of a rotational part, the production plan consists of a sequence of chucking and cutting operations by which the workpiece can be manufactured.

The general structure of the expert system which is being developed can be described by the model of expertise (Breuker & Wielinga, 1989) shown in Figure 1. From the concrete description of the workpiece and the available manufacturing environment more abstract feature descriptions are first constructed. These abstractions are then associated with an appropriate skeletal plan that has been stored in the knowledge base. The skeletal plan is finally refined with information from the workpiece and the factory description into the concrete production plan.

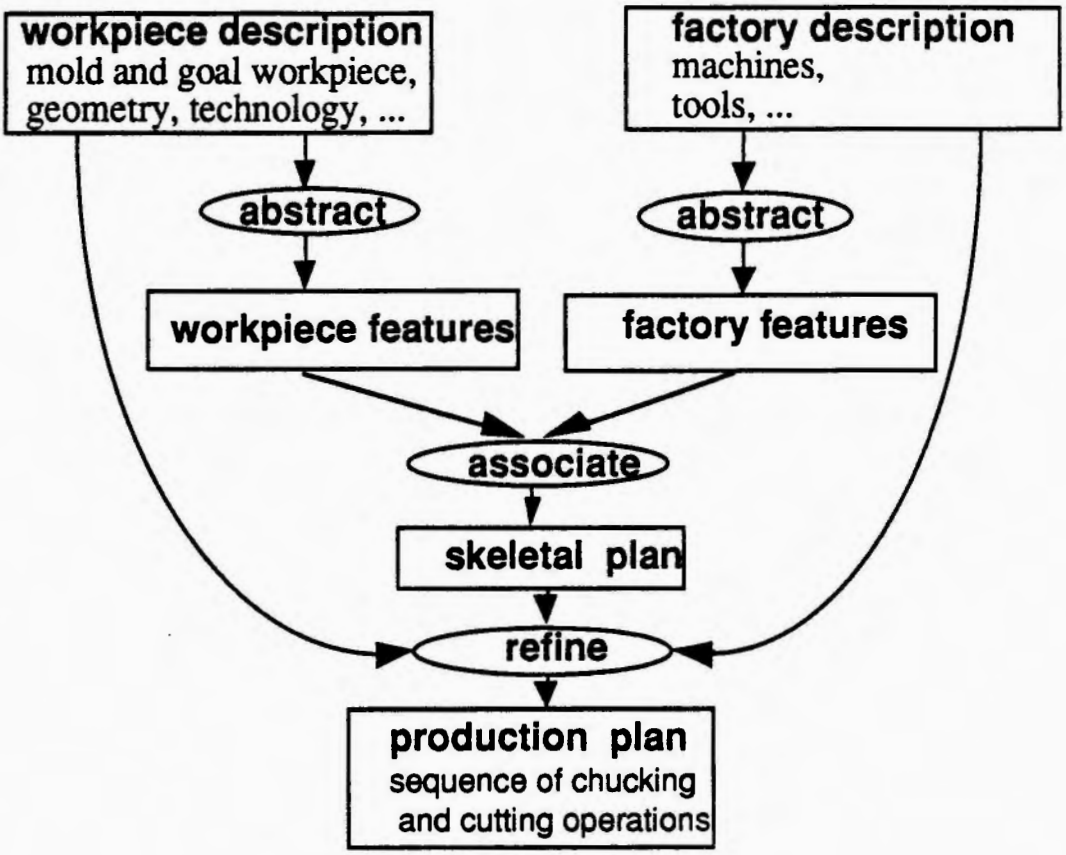


Figure 1: Model of expertise for production planning

The model of expertise specifies what kind of knowledge has to be acquired for the expert system, namely abstraction rules, refinement rules and skeletal plans which are associated with features of the problem description. In addition, a STRIPS-like model of mechanical engineering actions is presumed as a general domain model. This model requires chucking and cutting operations to be described by some typology and their preconditions and effects.

An integrated knowledge acquisition method is used to coordinate knowledge from texts, previously solved planning problems (cases) and the expert's respective memories. The knowledge acquisition tools COKAM (**C**ase-**O**riented **K**nowledge-**A**cquisition **M**ethod from Text) (Schmidt & Schmalhofer, 1990; Kuehn, Linster, & Schmidt, 1991) and CECoS (**C**ase-**E**xperience **Co**mbination **S**ystem) (Bergmann & Schmalhofer, 1991) are applied to the same set of cases so that the knowledge acquired with the two tools will complement one another. The domain and common sense knowledge supplied by COKAM and the definition of production classes obtained with CECoS, can then be utilized to automatically construct skeletal plans and associated application conditions through the explanation-based learning procedure SPGEN (**S**keletal **P**lan **Gen**eration Procedure).

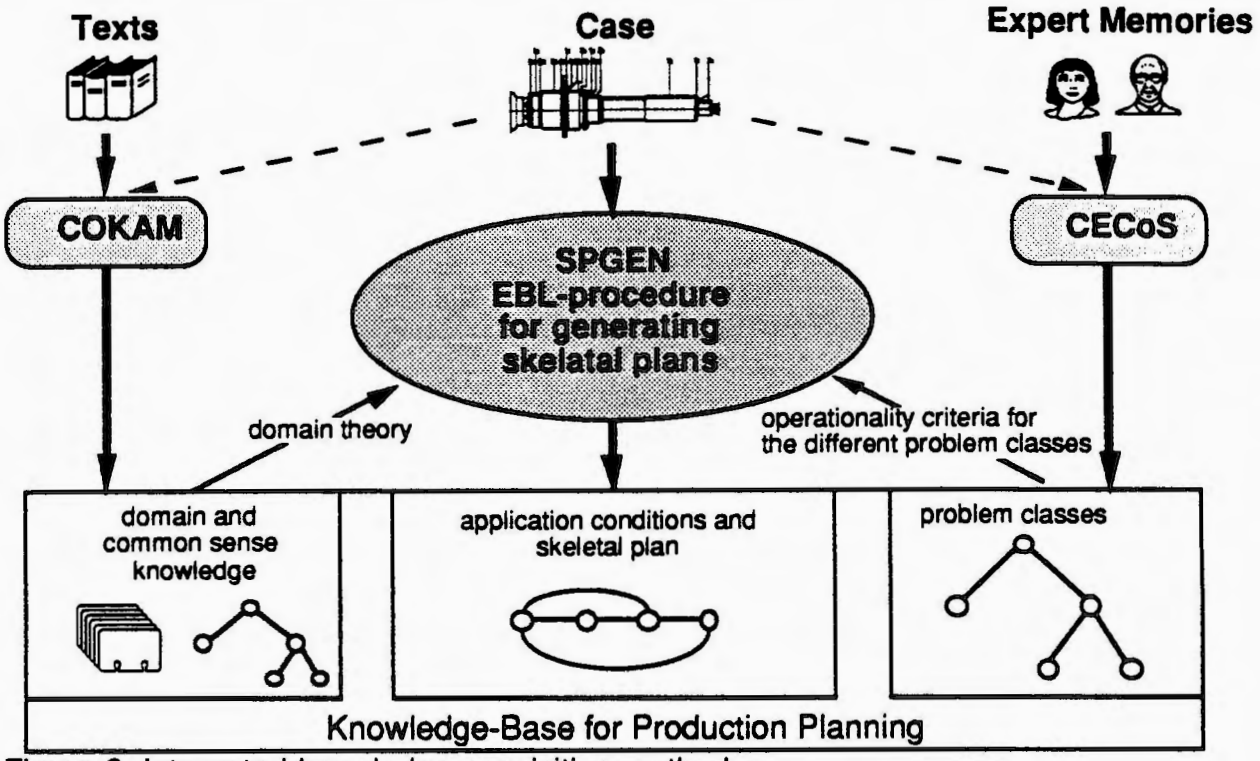


Figure 2: Integrated knowledge acquisition method

2.1. Case oriented knowledge acquisition with COKAM

With the interactive tool COKAM information is extracted from a text and subsequently enhanced by the expert's elaborations. The extracted information is then mapped to the model of mechanical engineering actions (domain model). The so collected knowledge thus provides an explanation of each step in the production plan and specifies the conditions which are required for its application and the resulting consequences. Table 1 shows a sample of text information and expert elaborations, which are relevant for determining the preconditions and consequences of a specific cutting operation. The mapping of the 3rd knowledge unit of Table 1 into the model of mechanical engineering actions, which will be described in section 3.1 shows that the extracted information needs to be properly interpreted.

1. For rough cutting the cutting speed should be 400 to 600 m/minute.
2. When the mold has been forged, bezeling is required, if ceramic cutting tools are to be used.
3. The surface roughness Rt depends on the cutting feed f and the corner radius r_e of the cutting tool and can be computed by the formula $R_t = f^2/8r_e$.
4. When thin workpieces are manufactured with a high cutting force, vibrations may occur.
5. When high tolerances are required, a very hard cutting material must be used for fine turning.

Table 1: A sample of text information extracted with COCAM

2.2. Acquisition of problem classes with CECoS

With the interactive tool CECoS a hierarchically structured set of problem classes is obtained from a set of prototypical cases and human expert judgements. The problem classes are defined so that a useful skeletal plan will exist for each problem class. From explicit and implicit memories, the expert first establishes an extensional definition of the various problem classes with respect to selected prototypical cases. The so established production classes are then intensionally and thereby generally defined.

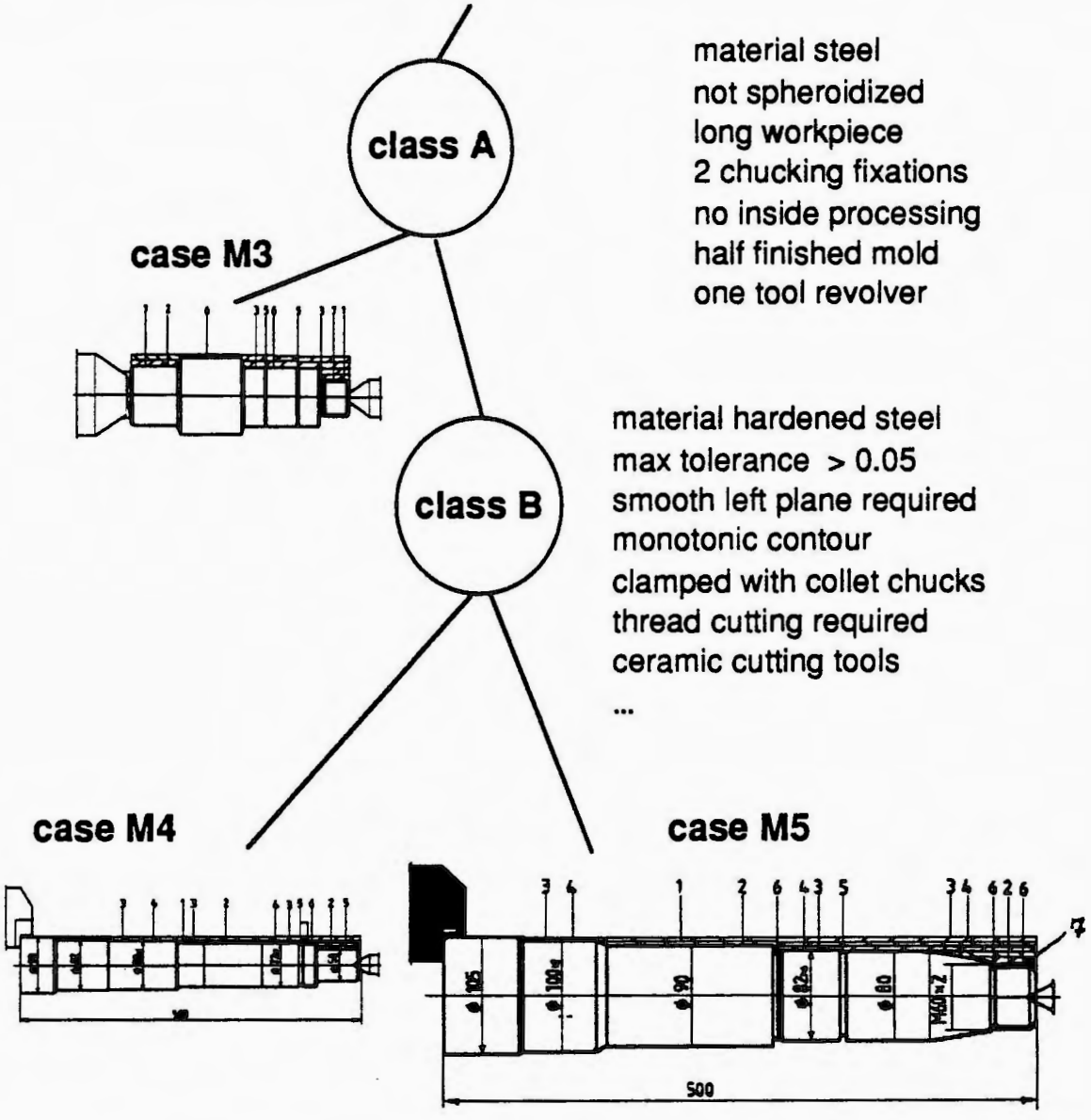


Figure 3: A section of a hierarchy of problem classes acquired with CECoS

Because the class definitions are based on expert judgements, the classes should be defined at the right level of generality: They should be general enough so that a large number of specific problems fall into the different classes and they should be specific enough to provide operational knowledge for production planning.

Figure 3 shows a section of the hierarchy of production classes which was obtained for some prototypical shafts. Class A is defined by the features which all three cases have in common. The more specific class B inherits all the features from class A and has some additional features which apply to the cases M5 and M4 but not to M3.

The features of each class may refer to the geometry (long workpiece) and technology (hardened steel) of the workpiece, to the factory (one tool revolver), or to the production plan (2 chucking fixations). As will be shown later, the features referring to the problem description (i.e. the workpiece or the factory) are utilized for the specification of the application conditions, whereas the features referring to the production plan are used for the definition of the operator classes in the skeletal plans.

3. PROCEDURE FOR GENERATING SKELETAL PLANS

SPGEN is based on explanation-based generalization as described by (Mitchell, Keller, & Kedar-Cabelli, 1986). The domain and common sense knowledge acquired with COKAM is thereby used as domain theory and the hierarchy of problem classes is employed to specify operationality criteria. Depending upon the selected problem class and the respective operationality criteria, a more or less general skeletal plan will be obtained from a given case.

A skeletal plan is constructed by SPGEN in four phases:

1. In the first phase the execution of the source plan is simulated and explanations for the effects of the individual operations are constructed.
2. In the second phase the generalization of these explanations is performed with respect to a criterion of operationality, that specifies the vocabulary for defining abstract operators for the skeletal plan.
3. In the third phase, a dependency analysis of the resulting operator effects unveils the substantial interactions of the concrete plan at the more general level of the skeletal plan.
4. In the forth phase the concept descriptions for the abstract operators of the skeletal plan are formed by collecting and normalizing the important constraints for each operation that were indicated by the dependencies.

For describing the SPGEN procedure we will use a simplification of the case M5 from Figure 3. The input and the (intermediate) results of the procedure will be presented in a PROLOG-like notation in which unquoted strings beginning with an upper-case character denote variables.

The formal representation of the case M5 which is used as input to SPGEN is shown in Table 2. The left side of the table shows the representation of the problem description which consists of the representation of the to be manufactured workpiece, the mold

and the factory. The geometry of the workpiece and the geometry of the mold is represented by elementary surfaces. The technology is represented by a specification of the tolerances, the material, the heat-treatment, etc. These specifications may apply to individual or to all surfaces. The production plan is represented as a sequence of chucking and cutting operations with various parameters.

Representation of problem description	**Representation of problem solution**
The workpiece Geometry surface(1, form(linear, (0,0), (0,52.5))), surface(200, form(linear, (0, 52.5), (60,52.5))), surface(201, form(linear, (60, 52.5), (67,45.5))), ... Technology tolerance(201, radial(0.05)), surface-finish(201,mean-roughness(0.01)), ... **The mold** Geometry surface(1, form(linear, (0,0), (0,52.5))), surface(2, form(linear, (0, 52.5), (500, 52.5))), surface(3, form(linear, (500, 52.5), ((500,3)))), surface(4, form(linear, (500,3), (497,0))), centerhole(40, 'Zen3mm', 1) Technology material(all, 'C45'), fabrication(all, half-finished), heat-treatment(all, none), ... **The factory** machine('PNE480'), no_of_tools('PNE480', 6), power('PNE480', 20 000), stiffness('PNE480', very-high), ...	**The production plan** operator(1, chuck(collet-chucks(15,soft), surface(2), force(200))), operator(2, cut(1, speed(450), feed(0.45), depth(5), form(linear, (66, 47.5), (500, 47.5)), tool(('FTC32-CSSNL3250-15', 'SNGN150816TO3030SN80')))), operator(3, cut(2, speed(450), ...)), ... operator(9, unchuck)

Table 2: Partial representation of a case used as input for SPGEN

3.1. Simulation and Explanation

In the first phase of SPGEN, the plan execution is simulated on the basis of the available domain theory. The simulation of the plan is performed by sequentially determining the effects of each operator $Op_1,...,Op_n$ of the plan. In order to determine the effects of the sequence of operators, the intermediate processing states from the initial state S_0 (the mold) to the final state S_n (which will contain the target workpiece if the domain theory is sufficient) are computed as follows:

$$S_0 \xrightarrow{Op_1} S_1 \xrightarrow{Op_2} S_2 \ldots S_{n-1} \xrightarrow{Op_n} S_n$$

The effects of the operator are represented by a set of rules with STRIPS like add- and delete actions. The execution of these rules thus create the successor world state. For example, knowledge unit 3 from table 1 is represented by the following rule:

IF operator(I,cut(speed(S),feed(F),depth(D),form(Form), tool(T)),
corner_radius(T,R),
produces_roughness(F,R,Roughness),
is_surface(Form,Surface),
THEN ADD(roughness(Surface,Roughness))

By applying all the rules for each operator, the various consequences of the individual operations of the plan are calculated. If the domain theory is sufficient, a complete explanation of the plan will be obtained. The proofs that exist for the applicability of each operator rule can now be seen as an explanation of each effect that depends on operator attributes as well as world state attributes, from the initial or intermediate states.

3.2. Generalization

In the second phase of the procedure, these proofs are independently generalized for each production step of the plan (explanation based generalization). The independent generalization of each production step is necessitated because of the complexity of the complete plans.

The degree of generalization is determined by the operationality criteria for each production step, which are defined at the concept (Hirsh, 1988) rather than at the predicate level. These criteria are obtained from the terms, which the texts and the expert used for describing the different operations of the concrete plan at a general level. It is thus assumed that exactly those terms which are used by experienced humans would determine operationality. A justification for this assumption can be found in the research of Rosch (Rosch, 1978): Rosch has shown that humans favor basic level categories in their descriptions. Such categories can be termed operational in the sense that they provide maximum information and the least cognitive effort for achieving some task goal.

3.3. Dependency Analysis

The dependency analysis of the third phase determines which previous operations (or initial state affairs) achieved the prerequisites for the various productions steps of the plan. It is thereby determined when the prerequisites for performing a specific production step were accomplished. A directed graph is constructed, in which all existing dependencies between the individual plan operations and the problem description are denoted by arcs. These problem descriptions, which were obtained through CECoS determine the generality of the skeletal plan to be constructed. The operationality criteria are provided by the features of the problem classes which were acquired from the human expert with the knowledge acquisition tool CECoS.

With the hierarchy of problem classes shown in figure 3, either the features of class B (and its subclasses) or the features of class A (and its subclasses) can be specified as being operational. In the first case a rather specific skeletal plan which applies to the problems of class B will be constructed, whereas in the latter case a more general skeletal plan for class A will be obtained.

Figure 4 shows a graphical representation of a part of the dependency graph that results from the analysis of the case M5. For example, cut 1 depends on the workpiece being chucked (for subsequent cuts this obvious dependency is no longer shown in the graph), on the geometry and the technology of the mold, and on the availability of ceramic cutting tools in the factory. It can also be seen from figure 4 that the first three cuts produce intermediate surfaces which are needed for the subsequent cut respectively but are no longer present in the final workpiece. The required geometry and technology of the goal workpiece is produced exclusively by the cuts 4 to 7, each of which produces some particular feature. The lack of a dependency between the cuts 5 to 7, furthermore indicates that they could be executed in any sequence.

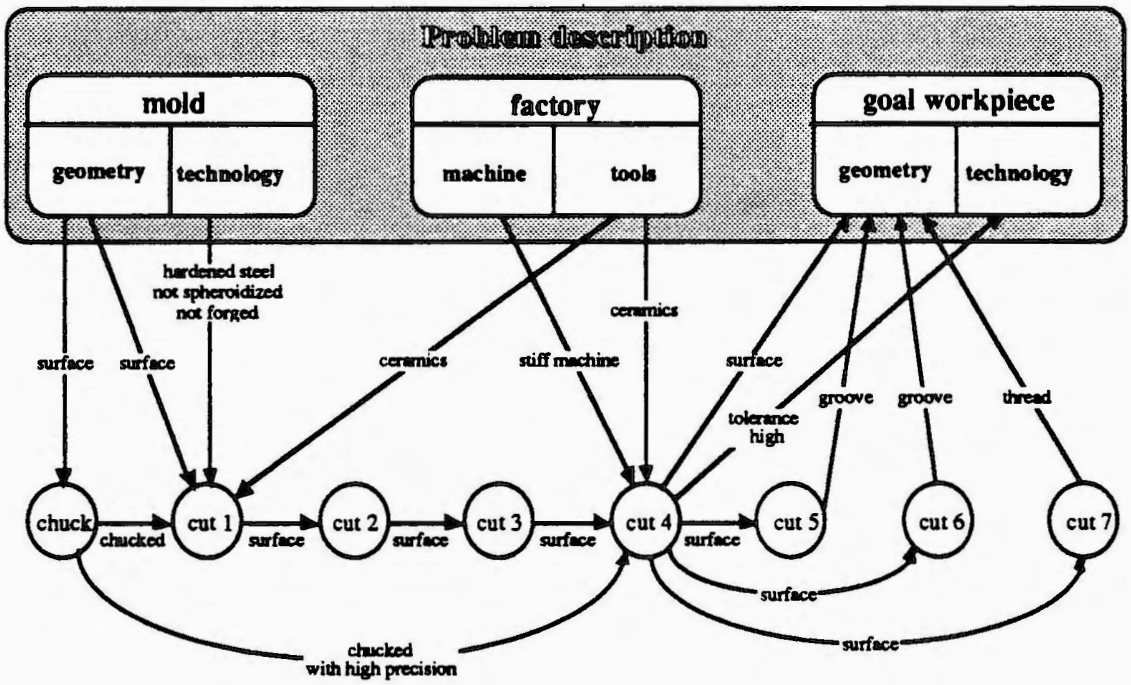


Figure 4: Partial dependency graph for the case M5

3.4. Normalization

This last phase builds the skeletal plan in its final representation by identifying independently solvable sub-formulas from the dependency graph which expresses only local constraints on one operator. By analyzing the occurrence of variables in the graph the dependencies are separated into:

- one set R_{Enable} that collects all dependencies that only relate to features of the problem description,
- one set R_{Opi} for each operator Op_i where the dependencies refer to parameters of the operator Op_i
- one set $R_{Dependent}$ where the dependencies refer to the possible orderings of the operator classes.

The set of constraints R_{Enable} formally describes the class of problems for which the skeletal plan can be used: it specifies the application conditions for the skeletal plan. The application conditions may refer to the mold, the goal workpiece or the manufacturing environment, as indicated in Figure 4. The skeletal plan itself consists of the set of operator classes $Op_1,...,Op_n$ with the constraints R_{Opi} and $R_{Dependent}$ which specify the possible sequences in which they may be applied.

For the case M5 the skeletal plan with application conditions shown in Table 3 is generated for the problem class B. The skeletal plan for the problem class A would be somewhat more general. For instance, it would allow any chucking tool with two fixations instead of collet chucks and would not require the material to be hardened steel.

Application conditions concerning	**The skeletal plan**
the workpiece Geometry surface(S1, form(linear, (0,0), (0,Z1))), 10 =< Z1 < 120, surface(S2, form(linear, (0, Z1), (X1,Z1))), ... Technology tolerance(S2, radial(Rt1)), Rt1 >= 0.025, surface-finish(201,mean-roughness(MR)), MR >= 0.01, ... **the mold** Geometry surface(Sm1, form(linear, (0,0), (0,Z1))), surface(Sm2, form(linear, (0, Z1), (X1, Z1))), surface(Sm3, form(linear, (X1,Z1), ((X1,Z2)))), surface(Sm4, form(linear, (X1,Z2), (X2,0))), centerhole(Sm5,Type,Depth), ... Technology material(all, Mat), mat_type(Mat,hardened_steel), fabrication(all, half-finished), heat-treatment(all, none), ... **the factory** machine(M), no_of_tools(M,N), N >= 6, power(M,P), P >= 15000, stiffness(M, very-high), ...	operator(1, chuck(collet-chucks(Width,soft), surface(S2), force(Force1))), 10 =< Width < 20, 200 =< Force1 < 300, ... operator(2, cut(speed(Speed1), feed(Feed1), depth(Depth1), form(linear, Star1,End1), tool(Tool1))), 400 < Speed1 < 600, 3 =< Feed1 < 5, 1 =< Depth1 < 6, cutting_material(Tool1,'SN80'), rake_angle(Tool1,45), tool_phase(Tool1,Phase1), 2 =< Phase1 < 3, ... operator(3, ...), ... operator(9,unchuck) **Dependencies** see bottom half of Figure 4

Table 3: Partial skeletal plan generated from case M5 for problem class A

A first version of SPGEN has been implemented in LPA-PROLOG on a MAC II computer [Bergmann90]. It can construct skeletal plans from simplified cases such as those shown in Figure 3. The current implementation deals mostly with the geometrical aspects and does not yet adequately take into account the technological and economical aspects of production planning.

4. DISCUSSION

The re-use of previously established solutions to hard problems has been suggested in the area of Artificial Intellegence (Riesbeck & Schank, 1989) as well as for software development in general (Fischer, 1987; Standish, 1984). In the area of Artificial Intelligence most approaches to the re-use of established solutions are discussed within the framework of case-based reasoning (Koehler, 1991). In case-based reasoning, the modification of an old case to a new problem is typically performed at the time when the new problem arises. By suggesting to systematically prepare sophisticated expert plans already during the knowledge acquisition process for an expert system these approaches are extended in the current paper.

Unlike case-based planning, the preparation of a case for its re-use is thus performed in ignorance of a specific new problem. It basically consists in analyzing and explaining a prototypical case in terms of a model of expertise and supplementary domain knowledge. Additionally, the features of problem classes which supposedly constitute the base level categories of human experts (Rosch, 1978) are used to determine operationality criteria for concepts in an explanation-based generalization procedure.

The skeletal plans and application conditions constructed with SPGEN, provide a combination of knowledge-based and heuristic abstractions of a concrete plan. For novel problems, which satisfy the application conditions, the skeletal plan will provide a knowledge-based partitioning of the novel problems into appropriate subproblems, which can then be solved more easily.

REFERENCES

Bergmann, R. (1990). *Generierung von Skelettplänen als Problem der Wissensakquisition.* Diplomarbeit, Universität Kaiserslautern, Kaiserslautern, Germany.

Bergmann, R., & Schmalhofer, F. (1991). CECoS: A case experience combination system for knowledge acquisition for expert systems. *Behavior Research Methods, Instruments, & Computers*, Vol. 23, 142-148.

Breuker, J., & Wielinga, B. (1989). Models of expertise in knowledge acquisition. In Guida, G., & Tasso, C. (Eds.), *Topics in expert system design, methodologies and tools* (pp. 265 - 295). Amsterdam, Netherlands: North Holland.

Fischer, G. (1987). Cognitive view of reuse and redesign. *IEEE Software*, 60-72.

Friedland, P.E., & Iwasaki, Y. (1985). The concept and implementation of skeletal plans. *Journal of Automated Reasoning*, Vol. 1, 161-208.

Georgeff, M.P. (1987). Planning. *Annual Review in Computing Science*, 359-400.

Hirsh, H. (1988). Reasoning about operationality for explanation-based learning. In *Proceedings of the 5th International Conference on Machine Learning* (pp. 214 - 220).

Koehler, J. (1991). *Approaches to the reuse of plan schemata in planning formalisms* (Technical Memo No. TN-91-01). Kaiserslautern, Germany: German Research Center for Artificial Intelligence.

Kuehn, O., Linster, M., & Schmidt, G. (1991, May). *Clamping, COKAM, KADS, and OMOS: The construction and operationalization of a KADS conceptual model.* To appear in Proceedings of EKAW '91 (Crieff) .

Mitchell, T.M., Keller, R.M., & Kedar-Cabelli, S.T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, Vol. 1, 47-80.

Riesbeck, C.K., & Schank, R.C. (1989). *Inside case-based reasoning.* Hillsdale, NJ: Lawrence Earlbaum.

Rosch, E. (1978). Principles of categorisation. In Rosch, E., & Lloyd, B. (Eds.), *Cognition and categorisation.* Hillsdale, NJ: Lawrence Erlbaum.

Schmalhofer, F., Kuehn, O., & Schmidt, G. (1991). Integrated knowledge acquisition from text, previously solved cases, and expert memories . *Applied Artificial Intelligence*, 5, 311-337.

Schmidt, G., & Schmalhofer, F. (1990). Case-oriented knowledge acquisition from texts. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., & van Someren, M. (Eds.), *Current trends in knowledge acquisition (EKAW '90)* (pp. 302-312). Amsterdam: IOS Press.

Spur, G. (1979). *Produktionstechnik im Wandel.* München: Carl Hanser Verlag.

Standish, T.S. (1984). An essay on software reuse. *IEEE Transactions on Software Engineering*, Vol. 10, 494-497.

Thoben, J., & Schmalhofer, F. (1990). *Wiederholungs- Varianten- und Neuplanung bei der Fertigung rotationssymmetrischer Teile* (Interner Bericht des ARC-TEC-Projektes). Kaiserslautern, Germany: German Research Center for Artificial Intelligence.

Studies in KADS Interpretation Models: Formalization, Linking to Domain Layers, and Tool Support

Wolfram Schmidt and Thomas Wetter
IBM Germany, Scientific Center
Institute for Knowledge Based Systems
Wilckensstr. 1a, D-6900 Heidelberg, FRG
EARN: WETTER at DHDIBM1

ABSTRACT

For the KADS knowledge acquisition method, interpretation models (IMs) are intended to guide and validate the transformation of verbal data into the conceptual model. Existing suggestions of IMs are limited in their usefulness due to lack of completeness, clarity, and precision. We present our further developments (cf. [Wett91a]) of denoting KADS IMs in a formal language and of offering them as part of a software tool for creating conceptual models. Formal notation guarantees an unambiguous meaning and enhances the usefulness of IMs for further formalization steps. We study generalizations that became evident from formalizing a number of IMs, illustrate the process of applying IMs in the modelling process, discuss the aspects of re-usability of elements of IMs and of how loosely or tightly coupled to real domains the notation of IMs can be, demonstrate the degree of overlap between the inference pattern of a real application and the best matching IM, and indicate how our formalization will be used in a future software tool.

1.0 INTRODUCTION

1.1 Conceptual modeling in KADS

After six years of ESPRIT funded research, KADS ([Wieli87]) has evolved into the most influential approach to knowledge engineering in Europe. Some of KADS' distinguishing features are that it aims at being domain independent (in contrast to domain related workbenches such as OPAL [Musen87] or KNACK [Klink88]) and independent of knowledge representation formalisms (in contrast to e.g. PRED [Xie88]).

It is central for the KADS approach that knowledge acquisition is a modelling activity. Different models serve different purposes in a waterfall like life cycle model of software production. Among the models used, the conceptual model plays a prominent role: From the perspective of knowledge acquisition as the first major phase, the conceptual model is the final document which is produced in a number of iterations and which fixes every identified item that was judged to be relevant for covering the contents of the application or the expertise to be modelled. From the perspective of system design i.e. the second major phase, the conceptual model is the mandatory input concerning the involved knowledge. In other words, the conceptual model is **the** interface between content related acquisition activities and artifact related design activities. (cf. [Wett90a]). This central role of the conceptual model has been ac-

knowledged by the KADS researchers by providing IMs as specific support for constructing conceptual models. Just to give a first idea: interpretation models exist for systematic diagnosis, monitoring, etc. ([Breuk87; Schre91]

The material presented subsequently is one of a number of attempts of stabilizing the KADS method by formalizing and operationalizing its components and processes.

1.2 The Role of Interpretation Models

Interpretation Models (IMs) are intended to support the construction of conceptual models. A conceptual model is organized in four layers:

domain	application specific static knowledge
inference	elementary application independent methods
task	major subproblems to be solved in the domain
strategy	control of solution process

I.e. the conceptual model describes an application by its specific **domain** characteristics in a bottom layer and applies **processes** of the three higher layers to this domain knowledge. The **inference** and the **task** layer are assumed to reflect general characteristics of inferencing.[1] This gave rise to the idea to describe them independently of specific domains and to use such abstract descriptions in order to understand and structure new applications.

These abstract descriptions have been termed interpretation models. They can play three significant roles in the KADS process of knowledge acquisition:

- They provide help and guidance to the knowledge engineer (KEngr). When facing verbal protocols or any other documentation from knowledge elicitation sessions, he has to give all items some meaning. In other words: he has to interpret. As we have outlined in [Wett91d], expert utterances can have such a variety of meanings that arbitrary interpretation may lead to arbitrarily unpredictable system behavior. The KADS answer to this is that problem solving can be understood as executing one out of a small number of stereotypes and that in the course of solving an individual problem the expert produces utterances which refer to elementary steps of that stereotype process that he applies. The stereotypes are the IMs, i.e. they can be understood as kind of frames, which supply slots where to place expert utterances.

- If we assume that for an area of application it is known which IM applies (and neglect the question how we could find that out), the IM plays a second role of validation. An item from the expert protocols is judged to have been correctly interpreted when it has found its place in the IM. Here comes one of the few places, where for our understanding KADS contains feedback: If a considerable number of items resist interpreting in the selected IM, the IM is put under question.

- If we furthermore assume that isolated IMs or their pure combination satisfactorily describe an essential set of applications, design elements can be created for the constituents of the IMs and the control expressions determining their application and re-used for different applications. This would emphasize the impor-

---

[1] Some KADS researchers also take the position that the task layer is application specific.

tance of IMs not only for the analysis and modeling phase but also for the subsequent design activities.

1.3 Status of Interpretation Model Research

The above arguments should make absolutely clear that IMs don't allow arbitrariness. A treatment of IMs has already been attempted in an early stage of KADS by [Breuk87]. The eleven authors of that text brought to bare their experiences from diverse fields of application and provided verbal, graphical, and semi-formal outlines of what they had arrived at as their abstractions of the encountered inferences. In this phase, a number of question marks still had to be placed in the text by the authors themselves, and readers found some more question marks. This may be one reason why the IMs have not been so much used in mainstream KADS experiments, whereas commercial users such as Bolesian™[2] have emphasized this aspect.

One reason for the question marks, in addition to the early stage of the whole endeavor, may have been that KADS did not at all prescribe a language for putting down the results from the phases. I.e. on one hand each KEngr could develop his personal style for IMs as well as for conceptual models. But on the other hand interpretation of the IMs and the conceptual models was somewhat arbitrary. The disadvantages of this arbitrariness for the conceptual models have in the meantime been recognized and led to different attempts of providing formalizations of the conceptual model ([Akker91; Fense91; Linst90; Wett90a]) and, recently, of interpretation models ([Wett91a]).

1.4 Outline of presentation

We now shortly resume the approach and report on evolving technical details. Next we consider the role of IMs in both research and application and demonstrate both by extended and commented examples. Finally we address aspects of a library of IMs such as reuse of constituents, coverage of real applications, variation of IMs, and the degree of coupling between models of inferencing and their respective models of static concepts.

2.0 INTRODUCTION TO THE LANGUAGE AND ITS FOUNDATION ON A GENERAL SORT MODEL

Many of the requirements for a formalization of KADS elements have been presented formerly in [Wett90a; Wett91a]. So we only give a short introduction including some internal of our research process concerning changes that happened while we worked out the details starting from [Wett90a].

Basicly [Wett90a] introduced a formal language which provides syntactic constructs for most constituents of the KADS domain, inference, and task layer and gives a semantic foundation of these language elements. The formal language resembles order sorted logic and adds some procedural elements.

---

[2] Bolesian is a trademark of Bolesian Systems Europe BV, Helmond, Netherlands

Concretely, for the **domain layer** we offer a lattice of sorts to represent concepts, subsumption, features of sorts and roles between sorts to impose structure upon the concepts, and we admit literals, universal quantification and axioms to express complex relations.

For the **inference layer** we allow literals (corresponding to a declarative knowledge base from which they might be proved) or procedure calls (corresponding to C- or other procedures importing algorithms where appropriate) as knowledge sources (KSs) to initiate basic inference steps. As will be outlined below, our method of dealing with metaclasses now also enables a conceptual, i.e. non-algorithmic description of many of the "bodies" carrying out the inference. For the time being we have abandonned the goal to support proof calls from within procedures; the full semantic foundation is harder than expected, and on the other hand we did not find definite need for deliberate nesting of procedural and declarative elements.

For the **task layer** we offer calls of subtasks and elementary subtasks realized by the above inferences, plus some control terms to express (what might be called knowledge level) control of the solution process. These include loops over sets of concepts, branching (IF-THEN-ELSE), since recently also consultation of sub-knowledge bases, and dynamic assertion of ground literals plus some support constructs (details see below).

To connect domain concepts to **roles** in the inferencing process represented as metaclasses of the KSs, we still use standard predicates *role_of*. But both our view of them and the technicalities of their use has considerable changed. Much detail will follow, but to give a first impression now: While formerly [Wett90a] we kind of applied *role-of(Domain-Var:Temporary-Role)*[3], we now prefer *role-of(domain-concept:structure-based-role)*[4]. This reflects our experience that working with IMs is enhanced by backing them up with highly general and hence flexible data structures, the general sort model (SM), whose elements "structure-based-role" can directly be used as metaclasses and whose structural restrictions further constrain the search space of making up a conceptual model (details see below).

One could think of the SM as kind of an inserted ceiling which is "below" the inference layer and next to the (future) domain layer and which allows for both: elegant formulation of tasks and KSs and easy linking to different domain structures.

A general SM is general in the following sense: Given a concrete IM such as assessment (cf [Breuk87]) the SM supplies the respective (data) structures that the inferences of the IM draw upon. In other words: there is a one-to-one correspondence between individual IMs and their respective SMs. Other IMs, e.g. systematic diagnosis, have different SMs.

This can be made use of in the process shortly described lateron of identifying the appropriate IM for a new application. The inference structure information is now backed up by domain structure information. Matching the stereotypical to the individual can now be based on process and static information.

One might think of a common generalization of the SMs attributed to the IMs as the virtual fully general SM. We did not make an attempt in this direction, because our

---

[3] Upper case indicates variables, i.e. dynamic entities.

[4] Lower case indicates sort or predicate name or sort relation, i.e. structural entities.

aim is not the most general structure for ontologies but pragmatic support of knowledge acquisition.

The following graph is intended to make the notation of SMs transparent using the SM of assessment as an example. It uses terms which are domain independent. The linking to an example domain will be outlined below.

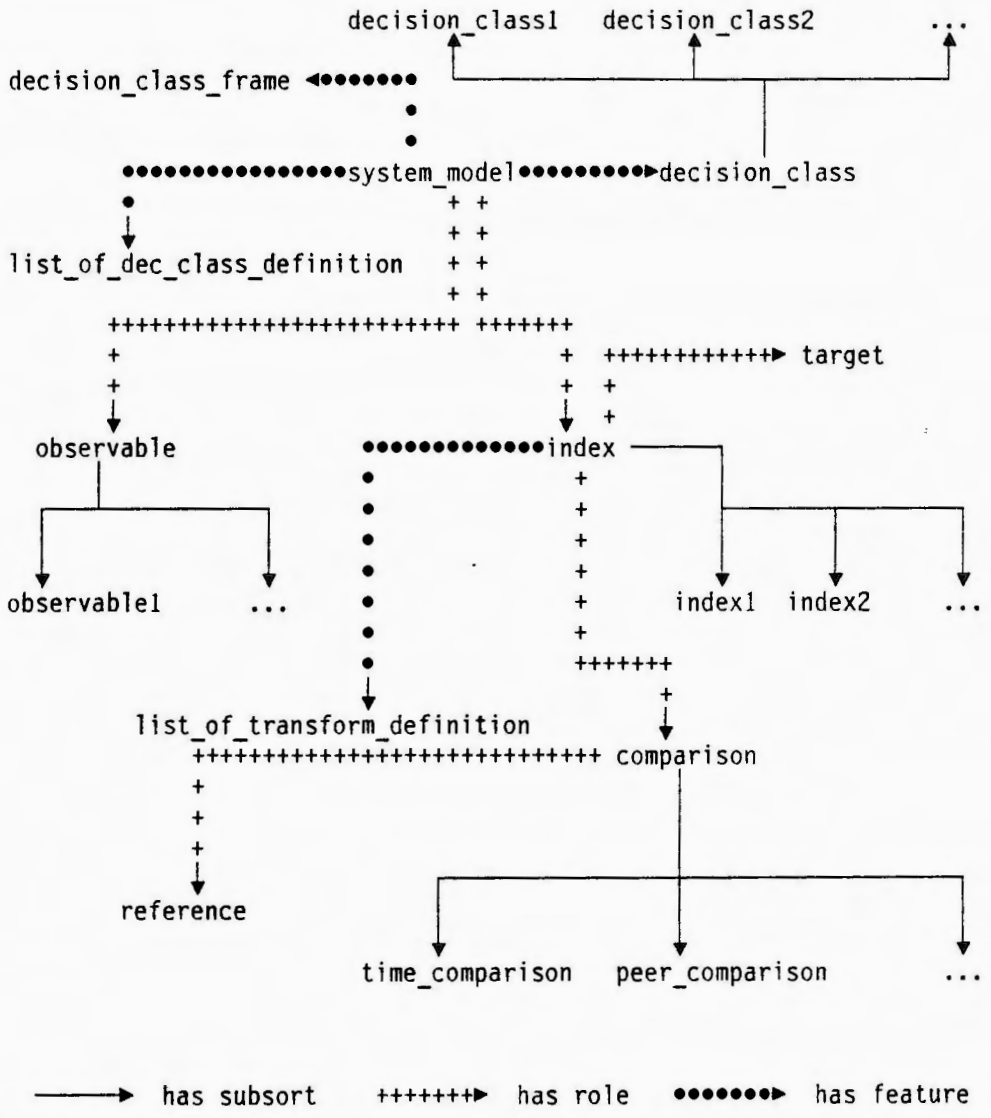


Fig.1: The General Sort Model For Assessment as part of the whole Sort Model

The **system_model** which identifies the to be treated part of the domain comprises structured knowledge and details of observation in its two roles:

- **Indices** respectively their subindices define the structured knowledge deemed to be general and in some form present in many applications. They are characterized by properties, formally denoted as *features*, e.g. a *list_of_transform_definition* to denote that complex application specific theories as well as simple computation formulae may be attached to the respective application independent notations. One might think of these lists as modules, but we prefer to call them list of axioms, because module is more part of a design

discourse than of a conceptualization discourse: modularisation involves aspects of maintaining consistency, efficient access, etc., whereas for conceptual modeling we need to express, what knowledge can be used for in which place of which inference.
Indices are connected to other items of structured knowledge; such a connection is formally denoted as a *role* which an item plays with respect to some other item in the structure. By the role *comparison_* the *target_* and *reference_* value of the index can be referred to.
It will be described below, what is needed to connect this general SM to specific sorts occurring in domains and what has to be added to cope with specific relations or dependencies of a given application.

- The state-description denoted by the subsorts of the sort **observable** can be transformed by adequate definitions into judgeable indices with their roles and features. That means that the observables are input for the transform definitions and the indices are their output. This is necessary if the underlying observed values can't directly be integrated into the structured sort hierarchy of indices. This may be due to a sort hierarchy written in abstract terms, which has to be applied to concrete, evtl. numeric values. The transformation process precedes the real examination in data driven tasks such as assessment.

3.0 NOTATIONS FOR IMS AND HOW THEY ARE INVOLVED IN PROCESSES

In the following we present two notations of IMs in detail and shortly comment on a third one. These notations are outcomes from processes and means of communication, whose understanding is a prerequisite to put the notations and their use in the right place. Notation 1 denotes the abstract inferencing pattern and makes references to its corresponding SM. Notation 2 adds links to concepts from a concrete application.

Notation 1 is a result from the research process. Its production and foundation requires the skill to create formal languages including their formal semantics (and in our case also of interpreters of sentences of the language in form of theorem provers.)

Recipient of this notation is the practicing KE. It is the responsibility of the scientist in his role of a sender to provide documentation and educational material i.e. providing the context in which the object level information can be understood.

The required level of understanding is that the KEngr is enabled to apply the notation to concrete applications, and eventually to slightly modify it in order to come up with a conceptual model of the application.

This involves the domain expert and the KEngr both in sender and recipient roles. The expert sends "traces" of his expertise, while the KEngr sends "traces" from his process of constructing the conceptual model. An indispensable second process in the KEngr is to make up the conceptual model. (And most probably as a side effect, the expertise of the expert undergoes a process of change during his attempts to send traces).

Let us assume for the moment that the constructive process in the KEngr operates as intended after sufficient education and supported by sufficient documentation. Then there still remains the problem that both KEngr and expert speak to each other in different languages each representing their respective areas of expertise. The expert uses domain terminology and the KEngr terms from the modelling approach. But in

the end they have to arrive at a document (named conceptual model) of which at least one of them has to take the responsibility to be an adequate conceptualization of the application under study.

Under normal conditions, there is no chance for a number of reasons to impose communication restrictions on the expert - we have to solve the problem given one sender resp. recipient with properties essentially out of our control. Hence we must enhance the KEngr both as a recipient and sender.

His recipient capacity might be called matching capacity, i.e. education and personality that enable him to detect structure (in our case: in terms of IMs and SMs) in the signals he receives. The patterns that he may find can be made more or less transparent by the first communication process, while the matching capacity is his, unless the researcher has also made such experiences and has communicated them. However, the research process has provided Notation 2 as a second language, which offers concrete means to express matches being detected.

The sender capacity of the KEngr is highly influenced by his personal insight in the meaning (semantics and procedural effect) of the expressions in the modelling language. The better he understands, the better can he communicate his understanding to the expert and eventually have it approved by the expert. This insight can be achieved by both education he receives in communication process 1 and experiences he makes. "Experiences" only occur in an operational sense, when the specification is executable, i.e. the KEngr sees behaviour traces of the model he builds.

Given all these considerations about processes involved we have to demand that the KEngr can produce instances of notation 2 representing individual applications he is modelling. Furthermore we have to demand either that he takes the responsibility for the notation arrived at or that he so clearly communicates it to the expert that the expert can approve it. These communication issues may be superimposed by contract conditions between a knowledge engineering company and their customer shifting the weight either towards the KEngr or to the domain expert.

Of all these processes and notations this text contains the following material:

- We introduce the language of Notation 1, i.e. a ***result*** of our research process and give some authentic detail about decisions and changes that happened during this process (Fig. 3).
- We include some documentational and educational material, however in a very superficial and preliminary form, to demonstrate by which means the first communication process can be supported (Fig. 2).
- We try to simulate the process of creating an individual Notation 2 for an (as will turn out) assessment problem (Fig. 4). This process has not really happened in the way subsequently outlined. We as researchers have analyzed and modelled the respective application before we had means of a formal language and the IM patterns phrased in that language. We have created the language and the IMs after the analysis, and now we look back how it might have happened given these new means.

3.1 Notation 1:

To support the matching activity of the KEngr we offer the "User's Guide" (of Fig. 2) which gives information about KSs and subtasks in form of their purpose, input and output, the method used to produce the output, their super- and subtasks and optionally notes for a more expressive description of the KS.

3.1.1 Elements used

Let us start with the explanation of some constructs which are used in the formalization:

- **USEALL_SORT** means a loop over a specific set of subsorts of a sort.
- **USE_AXIOMS** means use of additional specific axioms which are written down in specific knowledge bases. They are domain knowledge and are referred to by a pointer which is part of the SM called e.g. **list_of_transform_definition.**
- **Setof_sort** is a finite set of sorts.
- **identify_known(Setof_sort)** is a subtask which identifies a set of sorts where the members of this set are known, e.g. their values are available in the knowledge base or it is possible to compute them.

Furthermore we offer the KEngr a so called 'User's Guide' where he finds a brief information about the subtasks and KSs which are used in the IM. We present here a draft of the subtask *judge*:

purpose determine whether a candidate index deviates from its respective norm

input
- index to be checked
- standard against which to compare

output resulting difference

method retrieval of the required data items by means of the sort model and computation of their difference

subtasks
- a_select_target_value
- a_select_reference_value
- a_compute_difference

notes the subtasks are closely linked to the SM

Fig. 2: User's Guide for *judge*

3.1.2 Notation 1 in Detail

In the following we present the complete Notation 1 of the task assessment starting with three of its non-elementary subtasks and one elementary subtask.

```
judge(Index,Reference,Difference)
    ←
    Target_value : value;
    a_select_target_value(IN: Index;
                          OUT: Target_value);
    Reference_value : value;
    a_select_reference_value(IN: Index,Reference;
                             OUT: Reference_value);
    a_compute_difference(IN: Target_value,Reference_value;
                         OUT: Difference).
```

```
transform_state_description(System_model,Setof_known_index)
    ←
    initialize_by!(Setof_known_index,'EMPTY');
    identify_known(Setof_observable);
    identify(System_model,Setof_index);
    USEALL_SORT index IN Setof_index;
      Index : index;
      USE_AXIOMS transform_definition(System_model,Index)
                 WITH list_of_transform_definition_(Index);
        IF transform_definition(System_model,Index)
           THEN include!(index,Setof_known_index)
        END_IF
      END_USE_AXIOMS
    END_USEALL_SORT.

identify_known(Setof_reference)
    ←
    initialize_by!(Setof_referece,'EMPTY');
    USEALL_SORT reference;
      Reference : reference;
      IF known(Reference) THEN include!(reference,Setof_reference)
      END_IF
    END_USEALL_SORT.

a_select_reference_value(IN: Index,Reference;OUT: Reference_value)
    ←
    Index : index;
    Reference : reference;
    Comp : comparison;
    Reference_value : value;
    comparison_(Index,Comp);
    reference_(Comp,Reference);
    Reference_value=value_(Reference);

assessment(System_model,DCF,Decision_class)
    ←
    System_model : system_model;
    DCF : decision_class_frame;
    Decision_class : decision_class;
    transform_state_description(System_model,Setof_known_index); (α)
    USEALL_SORT index IN Setof_known_index;                      (β)
      Index : index;
      identify(Index,Setof_comparison);
      USEALL_SORT comparison IN Setof_comparison;                (γ)
        Comparison : comparison;
        USEALL_SORT reference;
          Reference : reference;
          IF judge(Index,Reference,Difference) THEN assert! (δ)
          END_IF
        END_USEALL_SORT
      END_USEALL_SORT
    END_USEALL_SORT                                              (ε)
    USE_AXIOMS dec_class_definition(System_model,Decision_class) (ζ)(η)
               WITH list_of_dec_class_definition_(DCF);
    END_USE_AXIOMS.
```

Fig. 3: Notation 1 of the IM assessment and its subtasks

3.1.3 Documentation of the Control Structure

The IM Assessment provides the attribution of a given case to a decision class. It proceeds as follows:

α Transformation of observed data using definitions which are domain specific. The definitions start out from the observed data and supply values about indices characterizing the domain, which can be e.g. target values, averages from peer groups, values from previous time periods, etc.
β Iteration over the set of indices
γ Iteration over the comparison bases
δ The actual index is judged, i.e. its referring comparison value is compared to the value based on the comparison base and the difference is asserted to the knowledge base in form of the construct "judge(Index,Reference,Difference)".
ε End of both iterations
ζ Iteration over definitions which determine decision classes according to the given decision class frame.
η Instantiation of these definitions with the asserted facts of the knowledge base "judge(Index,Reference,Difference)". The result is an attribution of the given case to a decision class, whereby the case is now presented in form of the judged indices.

[5]

3.2 The Process of Creating an Instance of Notation 2

3.2.1 Selecting an IM

In order to create a conceptual model by using (and evtl. modifying) an IM there are two essential steps

1. identifying the best matching IM
2. matching its expressions to details from the area of application

---

[5] The contents of the banking application in terms of Assessment might be presented to the domain expert in the following form:

α Iteration over definitions by which the balance sheet data can be transformed into max. 14 indices which characterize the profile of a customer referring to the judgement whether he is credit-worthy or not and transformation of the data into comparison values to which the target values of the indices can be compared.
β Iteration over the set of indices (if it was possible to derive all the indices from the balance sheet data then the set consists of 14 components, else a subset of them.).
γ Iteration over the set of comparison bases, i.e. in the first run-through the comparison base here is e.g. the standard, in the second the previou year, etc.
δ The target value of the actual index is compared to the value of the actual comparison base. The result is asserted to the knowledge base in form of the fact "judge(index,comparison_base,difference)".
ε End of both iterations
ζ Iteration over definitions determine decision classes according a given decision class frame, which can be the simple decision class "credit-worthy" or "not credit-worthy" ref. some more expressive decision classes between these two, such as "underwriter"
η Instantiation of the selected definitions with the results of the judgement. The output is an attribution of the given case to a decision class based on the judged indices.

This need not be a linear process, but we don't address the aspect of feedback and withdrawal of an IM here. We don't give detail about the selection process either but take the position that the decision tree of IMs in [Schre91] can be used.

Concretely, we assume that a first analysis has revealed that, given a company that is a customer of a bank and whose balance has to be analyzed to decide about the credibility, no construction or modification is required. Furthermore, the expectation is not to find a certain fault or to monitor the performance process but to have a single accept/reject decision for credit prolongation[6]. In [Schre91] this would be classified as assessment.

3.2.2 Matching domain and SM

In assessment as in all other IMs a decision has to be made about which entity in the area of application can be matched to *system_model* as the central term in the SM. In our case the company to be analyzed suggests itself, which gives rise to the first line in the instance of Notation 2 spelled out below:

```
role_of(company:system_model)
```

which is the explicit notation of a match that the knowledge engineer has arrived at. Next he may detect that assessment requires the set of ***possible*** decisions to be identified and what structure the ***individual*** decision has, giving rise to

```
role_of(decision_competence:decision_class_frame)
role_of(decision_about_company:decision_class)
```

This includes to match the inference transform_state_description to an inference occurring in the area of application. Here it can be matched to the activity to determine for which performance indices the balance sheet contains sufficient data for the value of the performance indices to be calculated. The respective ones are included in the set of known indices.

Let us leave for a moment the path of identifying domain structures and inferences each matching expressions found in the SM or IM. We insert a short outline of the coordinate operation of the IM and its link to a concrete domain layer. The following extract is part of the forthcoming sequence contained in the instance of Notation 2 which describes the balance analysis application.

```
role_of(company:system_model);
 ...
System_model : system_model;    (*)
 ...
transform_state_description(System_model,Setof_known_index) (*)
```

together which an extract from the definition of the subtask transform_state_description

```
role_of(balance_data:observable);
 ...
role_of(performance_index:index);
identify(System_model,Setof_index) (*)
 ...
```

is to demonstrate the following operation:

---

[6] The task of balance analysis was analyzed in cooperation with a research institute of a group of German banks based on the non-formal state of KADS. Deficiencies detected in this project stimulated our present research in formalization of KADS.

The IM, here represented by the lines (*) is preserved literally in the notation found in the IM library (unless minor mismatch is detected and the IM is adapted, see below), but its inferences, formulated in terms of the SM (system_model, index ...) are applied to concepts from the area of application (company, performance_index ...). This parameter passing or replacement mechanism is initiated by the *role_ofs* dotted in between the original lines from the IM. While for an IM its operation according to the intention can be assured off-line of a concrete application in the research process, its operation on the replaced parameters requires some consideration. Since most KSs don't treat concepts such as *system_model* in isolation but make use of the structures that *system_model* is part of, the requirement is that the structure to replace the terms from the SM must contain a homomorphic image (in a sense worked out elsewhere [Wett91c]) of the SM. In this case, inferences operational on the general SM will remain operational when applied to the individual domain structure.[7]

Getting back into the description of the modelling process now, we'll not continue exhaustively through all lines of Notation 2 but concentrate on few more details. They concern the "pointing" to domain theories (in contrast to domain structures that we treated above) to be used in inferences. *role_of(computation_formula:transform_definition)* denotes that in the case of analyzing the figures from a balance sheet the definitions to transform observations into indices have the form of computation formulae. The effect of this *role_of* is that in the IM we keep using the relation or predicate name *transform_definition*, which for a concrete inference in balance analysis is replaced by the predicate name *computation_formula*. This allows to make use of pre-existing conceptualizations or formalizations of the domain [8]. It furthermore allows to formulate application independent relations off-line and to import them into assessment and other processes by respective *role_ofs*. As a second related pointing operation we support

```
role_of(list_of_computation_formula_:list_of_transform_definition_)
USE_AXIOMS transform_definition(System_model,Index)
    WITH list_of_transform_definition_(Index)
```

with the following effect (also taking into account the replacement operations described above): *computation_formula(Company,Performance_index)* is evaluated by the theorem prover under additional use of axioms collected in a subtheory *list_of_computation_formula_*. Each *performance_index* "owns" such a subtheory, which can be pointed to by a feature *list_of_computation_formula_(Performance_index)*. This expression is also generated by replacement operations initiated by the role_ofs, provided that the feature *list_of_computation_formula_* is found in the structure of the domain in the very place where in the SM we had *list_of_transform_definition_*. The reader may take this as an illustration of the homomorphy requirement made above: When the concrete domain structure of balance sheets does not contain the respective feature *list_of_computation_formula_*, Notation 2 fails when executed.

---

[7] The fact that *role_ofs* can be dotted right in the middle of the pure IM seems to indicate that we have dynamic parameter replacement in our backminds: *role_of(performance_index:index)* may be used at some point in the inference and *role_of(something_else:index)* at a later time.
As a note from the research process: We don't really know at present what this means wrt the homomorphy claim just made.

[8] This definitely was an issue in the project we did with the group of banks: all the formulae had already been implemented before we started.

3.2.3 Adapting the IM

We conclude this simulation of a possible modelling process for the balance analysis domain with the hint to one of two changes that had to be made to the IM. The normal case in assessment is that who is in charge of an assessment knows his criteria. In the balance assessment we detected that statistical data in the form of branch averages or of former years' balance sheets may be lacking. This leads us to change the line

```
USEALL-SORT reference
```

of Notation 1 into

```
identify_known(Setof_reference)
USEALL_SORT reference IN Setof_reference
```

Primarily it can no longer be predicted that this is still an operational formulation - it has not been checked through in the research process but produced on the fly in the application process. However, *identify_known* occurs in a comparable constellation in the subtask *transform_state_description*, and *USEALL_SORT sortname IN Setof_sort* is a standard construct of the language and therefore is operational when *Setof_reference* is a finite set of sorts. Hence the expectation clearly is that this variant of the original IM is still within the scope of the procedural semantics provided in the research process.

3.2.4 The resulting conceptual model including one subtask

We present the full instance of Notation 2 forming the conceptual model of balance analysis, including one of its subtasks. Bold characters are from Notation 1, italic denotes replacement by domain concepts, bold italic deviates from the IM.

```
role_of(customer:system_model);
role_of(decision_competence:decision_class_frame);
role_of(decision_about_customer:decision_class);
assessment(System_model,DCF,Decision_class)
     ←
     System_model : system_model;
     DCF : decision_class_frame_;
     Decision_class : decision_class;
     transform_state_description(System_model,Setof_known_index);
     role_of(performance_index:index);
     USEALL_SORT index IN Setof_known_index;
       Index : index;
       role_of(balance_comparison:comparison);
       identify(Index,Setof_comparison);
       USEALL_SORT comparison IN Setof_comparison;
         Comparison : comparison;
         role_of(comparison_base:reference);
         identify_known(Setof_reference);
         USEALL_SORT reference IN Setof_reference;
           Reference : reference;
           IF judge(Index,Reference,Difference) THEN assert! END_IF
         END_USEALL_SORT
       END_USEALL_SORT
   END_USEALL_SORT
   role_of(judgement:dec_class_definition);
   role_of(list_of_judgements_:list_of_dec_class_definition_);
   USE_AXIOMS dec_class_definition(System_model,Decision_class)
            WITH list_of_dec_class_definition_(DCF);
   END_USE_AXIOMS.
```

```
transform_state_description(System_model,Setof_known_index)
    ←
    initialize_by!(Setof_known_index,'EMPTY');
    role_of(balance_data:observable);
    identify_known(Setof_observable);
    role_of(performance_index:index);
    identify(System_model,Setof_index);
    USEALL_SORT index IN Setof_index;
      Index : index;
      role_of(computation_formula:transform_definition);
      role_of(list_of_computation_formula_:
              list_of_transform_definition_);
      USE_AXIOMS transform_definition(System_model,Index)
                 WITH list_of_transform_definition_(Index);
        IF transform_definition(System_model,Index)
           THEN include!(index,Setof_known_index)
        END_IF
      END_USE_AXIOMS
    END_USEALL_SORT.
```

Fig. 4: Notation 2 of the IM assessment and the subtask transform_state_description

3.3 Notation 3:

Notation 3 does not exist independently but is the application of Notation 2 during consultation of the system. Automatically the *role_ofs* are instantiated as

role_of(instantiation_of_domain_concept:structure_based_role)

In a special application the user of the system gives and receives instantiations of the sorts for one single case. He need not know anything of the general SM or how the processes inside the model are realized, but can understand how the IM works, too.

To summarize: we built one model and produce two different descriptions of this model which are intended to specifically support the process outlined above.

3.4 Variation among IMs

To give idea of the degree of similarity or dissimilarity of different IMs, we present a different IM, namely "systematic diagnosis" as pure code (without the just given subtask *judge*) and shortly hint to characteristic differences. The following presentation of the IM systematic diagnosis is in the form of Notation 1:

```
systematic_diagnosis(System_model)
    ←
    System_model : system_model;
    Index : index;
    a_select_index(IN: System_model
                   OUT: Index);
    examine(Index).
```

```
examine(Index)
    ←
    USEALL_SORT index;
      Subindex : index;
      initialize_by!(Comparison_base,standard);
      judge(Subindex,Comparison_base,Difference);
      If (Difference<>0) THEN examine(Subindex) END_IF
    END_USEALL_SORT.
```

Fig. 5: Notation 1 of the IM systematic diagnosis and the subtask examine

Every IM treats one specific part of the general SM. Comparing the SM of the IM "systematic diagnosis" with that of "assessment" we see that in the first one there is no sort *observable*. In "systematic diagnosis" we directly address the faulty component of a system in form of an *index* and search for the specific fault of this component, i.e. we want to know the final most precise specification of it in form of a *subindex* or *subsubindex*, etc.. The knowledge used is just given in form of indices. There is no necessity for a preceding transformation process from observable data to structured data like in "assessment". The examination in "systematic diagnosis" is straight forward, i.e. when the most specific fault is found the process ends, whereas in "assessment" all possible judgements about all computable indices are carried out and the result is a *decision_class* which is deduced from all these judgements.

Another difference is the recursive structure of "systematic diagnosis". Because we go through the hierarchy of *indices* and *subindices*, i.e. from less specific information to the desired specific one, we need a recursive call of the subtask *examine*. The *index* is successively decomposed into its subindices and by a loop over subindices of one level they are judged whether there is a difference between their *target* and the *reference* value which represents the standard norm. If so there is a recursive call for *examine* and the *subindex* is further decomposed and so on.
In "assessment" there is no recursive structure because the desired information is derived from a flat structure of indices and their deviations from norms which are given as several comparison bases. Further structural information of "assessment" resides in the relations between *observables* and *indices*, as outlined in the presentation of *list_of_computation_formula_* of the banking application.
On the other hand we see that we can use the same process of judging an index in two different IMs beginning from different assumptions and receiving different kinds of results.

4.0 USE OF A LIBRARY OF FORMALIZED IMS

4.1 Observations of Usability

The critics about IMs are that there is no end in the process of building new IMs and that it is not possible to cover a large application with the existing IMs ([MLN91]). There will always rest a certain part of uncovered knowledge because of the complexity of every application. In our example this is manifest by section 3.2.3.

- ***identify_known(setof_reference)*** introduces an additional inference required in our sample banking application. While the "standard" assesssment assumes that comparison values for the indices supporting the judgement are known, the banking example indicated that the statistics supplying the respective data may

be lacking in some cases. Hence the conceptual model can only iterate over those indices, whose existence of values has been checked before.

- Another example of deviation for which we don't supply the code here was that companies from different branches (manufacturing, trading etc.) and sizes are judged on the basis of different indices.

Such examples indicate the limited precision and coverage of fixed IMs. But there are still advantages of IMs when a tool to use them is offered:

- It needs less time for the KEngr to detect whether the application might be structured in the way for using it afterwards as a knowledge based system.

- Even if the whole domain couldn't be built up in the system, a large part of it is just well-structured after the knowledge acquisition phase supported by IMs and we offer the KEngr the possibility to develop his own variant of an IM using the just implemented KSs to put the rest of the domain into the system. After having experiences in using pre-existing IMs it might be easier for the KEngr to construct such an IM by his own using facilities such as a library of existing KSs and subtasks with attached descriptions, the available control instruments, etc.. If he wants to use one KS which is not yet available he will be in the position to specify the input and output of this KS, and when an external (C-) procedure seems appropriate, can give it to someone for implementation and finally to integrate it into the IM. In case of procedural constructs inside the realm of our language he can produce the KS himself, as exemplified by *a_select_reference_value* in section 3.1.2.

- The most important fact which results from our work in implementing IMs is that we can demonstrate the reusability of KSs and subtasks in IMs. The more IMs we have implemented the less work was to be done in implementing new constituents. It is obvious that the process of creating new KSs and subtasks converges and therefore we hope that after a finite number of implemented IMs nearly all of the KSs exist which might be needed for further applications. A consequence of this hypothesis would be that the KEngr has the disposal over all the KSs which he needs to built new IMs for a special application. No more phrase work would have to be done. This means that new applications can be constructed in a short time and cause less costs.
  As confirmation of our statement we demonstrate the reusability of KSs by the table on the next page.

- By the time using the IMs the KEngr becomes familiar with those constituents and hence will be able to construct a model of the expertise without much looking at the descriptions of the KSs and subtasks in form of their purpose, methods, input, output, sub- and super-tasks. The time required for modeling a new application will be decreasing.

knowledge source	systematic diagnosis	heuristic classification		monitoring		assessment	prediction
		data driven	solution driven	data driven	model driven		of behavior
Elementary-KSs	*use of the structure of the SM; elementary computation*						
a_select_index	x						
a_select_reference_value	x			x	x	x	x
a_select_target_value	x			x	x	x	x
a_select_tolerance				x	x		
c_pattern_match					x		
a_compute_difference	x			x	x	x	x
Test-Control-Subtasks	*modifying variables supporting the control flow*						
identify_known		x	x	x	x	x	x
match		x					
match_inverse			x				
select_from_hypothesis					x		
execute_relation							x
transform_into_state_description			x				x
transform_state_description		x				x	x
Elementary-Control-Subtasks	*mechanical manipulation of variables supporting the control flow*						
include!		x	x	x	x	x	x
initialize_by!	x	x	x		x	x	x
unite!							x
Navigation-Subtasks	*climbing or descending in the hierarchy of the sort model*						
identify		x	x		x	x	x
identify_inverse		x					
Update-Subtasks	*updating the knowledge base*						
specify_discrepancy				x	x		
assert!			x	x	x	x	x
Test-Subtasks	*test and pursue deviation from norm*						
examine	x						
Compute-Subtasks	*computation of values*						
judge	x			x	x	x	x

Table 1. Reusability of Knowledge Sources in IMs

4.2 Status

Implementation of IMs as part of a larger project to provide a modeling tool is being done on IBM PS/2 under AIX, using Quintus Prolog and C.

5.0 Discussion

Interestingly, this endeavour of formalizing a number of IMs, which may appear as a funny exercise at first sight, not just brought three formal notations of entities that had already been suggested in informal notation. Furthermore the way we did the formalization supplied additional insights about the nature of the IMs which are related to the tightness of coupling to real domains and to the degree of re-usability of constituents (KSs and subtasks). A first attempt of rigorously applying a formalized IM to a real application clearly indicated the scope and coverage of the IM guided procedure.

The first and conceptually most prominent observation was that the primary need to construct the general SM as a foundation of the "empty" inferences which resemble the introduction of formal parameters lead to the insight that the terms (sort names, features, etc.) that we tended to use in the SM correspond to dynamic role names that we had been using in [Wett91a] as dynamic role attribution. As a consequence we ended up with a number of redundant role declarations such as *role-of(Reference,Reference)*, with the first parameter to denote a sort name in the SM and the second to denote a metaclass to be used by an inference. So we decided to cancel the role-of construct from the Notation 1 of IMs and to reserve it to the linking of domains, i.e. for *role_of(domain_expression;sort_name_of_general_sort_model)* with the normal case that *sort_name_of_general_sort_model* not only reflects domain structure but also role in inferencing. Technically this is a minor change and eases reading of IMs. Conceptually it tends to indicate that formalizing (even the highest abstraction of) domains happens with inference in mind and vice versa. In other words: Additional reason for a slight move away from the KADS I paradigm of independence and free recombination of domain related knowledge and inference knowledge towards the KADS II attitude of more adhering to the interaction hypothesis arises from formalization attempts.

Technically (and evtl. conceptually, if one thinks of building blocks of problem solving) most interesting is the aspect that during the construction of 7 IMs, which is about a third of the IMs suggested in earlier KADS publications ([Breuk87]), the number of KSs and subtasks required to cover the inferencing needs of tasks and subtasks only increased in the beginning and seemed to converge once some IMs had been formalized. It is certainly too early to call these generic building blocks, also because of the partially rather technical nature of some of the knowledge sources as presented in this text, but it seems worth further investigation.

This is the more so, because if this or a modified future surprisingly small set of elementary functions covers a major part of inferences required for KBS, then supplying design elements for these few functions would essentially support the integration of knowledge engineering in the subsequent phases of software engineering ([Wett91b]).

Unfortunately, there is contrary evidence. In a number of discussions the opinion or observation was expressed that IMs serve an important purpose as mental help for the KEngr, but they tend not to be sufficiently flexible to fully cover individual applications. Our own first attempt to apply our approach revealed that

1. among the IMs available assessment best matched a balance analysis application we had been facing

2. assessment offered some facet going beyond the need of our customer (they could do with a fixed small set of decision classes), i.e. in this sense the domain was homomorphic to the general sort model, as required above.

but

3. assessment expected reference values to be available
4. assessment did not offer axioms for final dec-class-definition dependant of sort-of-system-model (in this case branch).

Obviously, they could have been added without problem (and indeed we demonstrated that the conceptual model contains these faces) but this would change the role of IMs from fixed building blocks towards raw material, from which the final conceptual model can be constructed as a variant. Pragmatically, this may be recommended and may even be the tacit procedure in many application of KADS. But it puts under questions whether

- IMs can still be used for validating items from the linguistic data, if they themselves undergo change and criticism
- there remains any use of supplying design elements for the constituents of IMs, facing the risk that just those elements equipped with elegant, efficient, etc. design elements will be modified in such a way that they are no longer covered by the design elements.

In our opinion the SM can be understood independently of the inference because on the one hand a KADS trained KEngr is familiar with the used sort names and on the other hand when it is put in concrete domain expressions it is intelligible for the expert. The hierarchical structure (e.g. subindices of indices) and the possibility to attach properties of sorts and to define their roles is close to the intuitive imagination of a domain. Therefore it may cover the requirements of most applications.
There is no concrete experience in using notation and method in more complex applications. Optimistically we can point to the possibility to change IMs and even to create conceptual models fully independently of any existing library of IMs. However, the language used puts up actual and definite limits to what can be expressed. Actual ones concern that modifications of the sort model are not supported, i.e. constructive processes cannot be modelled. The underlying language (Llilog) would support these operations, and hence this limitation can be removed or reduced in later versions. Principle ones concern probability based inferences and other types of non logic-based inferencing or full treatment of temporal reasoning or other "overlays" to logical inferencing which are highly expressive on their own and would compete with the expressiveness of our language of using sorts and procedures. These principle limitations can partly be overcome if inferences as the ones just mentioned can be exported into a treatment by attached procedures, i.e. if the whole problem can be factorized.

REFERENCES

[Akker91] **Akkermans, H., van Harmelen, F., Schreiber, G., Wielinga, B.** *A Formalization of Knowledge-Level Models for Knowledge Acquisition* accepted for Int. J. Intelligent Systems

[Breuk87] **Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billault, J.-P., Davoodi, M., Hayward, S.** *Model-Driven Knowledge Acquisition: Interpretation Models* ESPRIT Project 1098, Deliverable task A1, U. Amsterdam, Dept. Social Science Informatics, 1987

[Fense91] **Fensel, D., Angele, J., Landes, D.** *Knowledge Acquisition and Representation Language (KARL)* Expert Systems & their Applications, Proceedings Avignon, 1991

[Klink88] **Klinker, G., Boyd, C., Dong, D., Maimann, J., McDermott, J., and Schnelbach, R.** *Building Expert Systems with KNACK* in: Boose, J. and Gaines, B. (eds.); Proc. 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff (Canada) Nov. 1988

[Linst90] **Linster, M.** *Declarative Problem Solving Procedures as a Basis for Knowledge Acquisition: A First Proposal* in: Worksheets of the GMD, June 1990

[Musen87] **Musen, M.A., Fagan, L.M., Combs, D.M., and Shortliffe, E.H.** *Use of a Domain Model to Drive an Interactive Knowledge-Editing Tool* Int. J. Man-Machine Studies **26**, (1987), 105-121

[MLN91] **Mütter, S., Linster, M., Neubert, S.** *Handouts of KADS-Users-Group-Meeting* Voß, A., Ueberreiter, B. (eds), München, February 1991, SIEMENS Technical Reports KI

[Schre91] **Schreiber, A., Wielinga, B., Breuker, J.** *The KADS Framework for Modelling Expertise* Proc. EKAW91, European Knowledge Acquisition Workshop, UK, May 1991, (forthcoming) Springer

[Wett90a] **Wetter, Th.** *First order logic foundation of the KADS conceptual model* in: Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and Van Someren, M. (eds.), Current Trends in Knowledge Acquisition, Amsterdam: IOS Press (1990)

[Wett91a] **Wetter, Th. and Schmidt, W.** *Formalization of KADS Interpretation Models* in Steels, L., Smith, B. (eds) AISB91, Artificial Intelligence and Simulation of Behaviour, Springer Verlag London, 1991

[Wett91b] **Wetter, Th.** *Knowledge Engineering vs Software Engineering - Old Wheels or New Aims and Methods?* Proc. EKAW91, European Knowledge Acquisition Workshop, UK, May 1991, (forthcoming) Springer

[Wett91c] **Wetter, Th.** *Formalisation and Tool Support for KADS Conceptual and Interpretation Models (working title)* manuscript in preparation

[Wett91d] **Wetter, Th. and Nüse, R.** *Use of Natural Language for Knowledge Acquisition - Strategies to Cope with Semantic and Pragmatic Variation* accepted for IBM J. Res. Dev.

[Wieli87] **Wielinga, B.J., Bredeweg, B., and Breuker, J.A.** *Knowledge Acquisition for Expert Systems* Nossum, T. (ed.); Advanced Topics in Artificial Intelligence; Lecture Notes in Artificial Intelligence 345, Berlin 1987 (Springer), 96 - 124

[Xie88] **Xie, S., Dumaresq, D.F., and Winne, P.H.** *PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages* in: Boose, J. and Gaines, B. (eds.); Proc. 3rd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop; Banff (Canada) Nov. 1988

Knowledge Acquisition for Temporal-Abstraction Mechanisms

Yuval Shahar, Samson W. Tu, Mark A. Musen

Medical Computer Science Group, Knowledge Systems Laboratory
Departments of Medicine and Computer Science
Stanford University School of Medicine
Stanford, California 94305–5479
Email: shahar@camis.stanford.edu

ABSTRACT

We describe three general temporal-abstraction mechanisms needed for managing time-stamped data: point temporal abstraction (a mechanism for abstracting several parameter values into one class); temporal inference (a mechanism for inferring sound logical conclusions over a single interval or two meeting intervals); and temporal interpolation (a mechanism for bridging non-meeting temporal intervals). Making explicit the knowledge required for temporal abstractions supports the acquisition of problem-solving knowledge needed for planning, plan-execution, problem identification, and plan revision. These mechanisms are implemented in the RÉSUMÉ system, and will be used in the context of our ongoing PROTÉGÉ–II project, whose goal is to generate automatically knowledge-based systems, as well as the appropriate knowledge-acquisition tools, custom-tailored to acquire the specific domain and task knowledge needed by the specific problem-solving method chosen for the task.

1. INTRODUCTION

A task common to many application domains is the analysis of data accumulated over time, leading to identification of past and present trends and to episodic decisions made on the basis of the previous and current states of the world. An example of such a task in the medical domain is managing patients being treated under clinical protocols. An inherent requirement of the task of clinical-protocol management is to accumulate and analyze patient data over time, and constantly to revise an assessment of the patient's condition by abstracting higher-level features and states (e.g., a drug-toxicity state) from raw numerical data (e.g., white-blood-cell [WBC] counts). These higher-level features can be used in monitoring the effects of the interventions indicated by the protocol, in recognizing problems, and in suggesting possible modifications to the original treatment plan. Knowledge-based systems can assist physicians in the initial plan-formation phase, as well as in the problem-identification and plan-revision stages.

During the 1980s, our group developed a series of knowledge-based systems and knowledge-acquisition (KA) tools for managing patients who are enrolled in clinical trials using protocols (Tu et al., 1989; Musen et al., 1987; Musen, 1989). The ONCOCIN system assists physicians caring for patients with cancer who are being treated under clinical protocols (Tu et al., 1989). OPAL

(Musen et al., 1987) is a knowledge-acquisition tool for ONCOCIN that acquires knowledge about specific oncology protocols from domain experts. The e-ONCOCIN system is a domain-independent shell based on the problem-solving method of episodic skeletal-plan refinement embodied in ONCOCIN (Tu et al., 1989; Musen, 1989). The e-ONCOCIN system's temporal database represents as time intervals only planning entities (e.g., drug treatments); clinical abstractions can be attached only to specific time-stamped points. However, creating new, interval-based abstractions from temporally oriented data is a process relevant to most clinical-management contexts. Caring for patients being managed under protocols for treating hypertension or AIDS, for instance, requires recognition of time intervals during which the blood pressure was abstracted as STABLE or during which the WBC count was abstracted as DECREASING. Specifying *explicitly* the temporal-abstraction mechanisms and the *knowledge* required to instantiate those mechanisms for any task is advantageous for three reasons:

1. Knowledge of the parameters needed to abstract higher-level concepts from numerical data and from intermediate-level concepts enables acquisition of these parameters from domain experts (e.g., expert physicians) and instantiation of domain-independent temporal-abstraction mechanisms.

2. The temporal-abstraction mechanisms support recognition of dynamically changing states and problems, suggestion of an initial plan, assessment of the effects of the current plan, and modification of existing plans during execution time.

3. Explicit temporal-abstraction knowledge used in the context of a time-oriented database enables reasoning about time-oriented records over time, independent of consultation sessions with the user. The system can update its conclusions periodically or whenever new data arrive, and can present them independently of its recommendations. For instance, a knowledge-based planning system can (at least conceptually) separate its reasoning for identifying problems and goals from its reasoning for planning and execution

2. THE METHOD-CONFIGURATION PARADIGM

We are developing mechanisms for temporal abstraction within the framework of the PROTÉGÉ–II project (Tu et al., 1991; Puerta et al., 1991). PROTÉGÉ–II is a system designed to overcome some of the limitations of the original PROTÉGÉ system, such as adherence to a specific problem-solving method (i.e., skeletal-plan refinement). PROTÉGÉ–II generates knowledge-acquisition tools for multiple tasks (e.g., clinical management or diagnosis) and for multiple domains (e.g., oncology or management of AIDS patients), using different problem-solving methods. Each tool is specific to the domain, task, problem-solving method, and temporal model defined by the PROTÉGÉ–II user. A corresponding custom-tailored problem solver is generated in parallel. In the case of the skeletal-plan–refinement method, the oncology domain, and the protocol-management task, this paradigm would generate both OPAL and e-ONCOCIN. Our basic approach relies on the assumption that the design of a knowledge-based system requires relatively few high-level problem-solving methods. For example, the skeletal-plan–refinement method can be recast as one of the family of the "Propose [a plan], Critique [the proposed solution], and Modify [and revise the plan accordingly]" (PCM) methods (Chandrasekaran, 1990). The "Generate, Test, and Debug" paradigm (Simmons, 1988) is another example of a PCM method—in this case, for an interpretation task. We are developing a library of such reusable, domain-independent, problem-solving methods. These methods will be indexed by the tasks that use them. A *task* includes a set of inputs and outputs of certain characteristics and a set of semantic constraints on the relationships between the inputs and the outputs with a specific underlying

structure. An example of a task is "recommend episodically a set of appropriate actions, given a patient's time-oriented record and a set of protocols and protocol-management knowledge" (Figure 1). A task can usually be accomplished by several possible *methods*, which it indexes. Methods typically decompose tasks into several subtasks, with no commitment regarding the subsequent methods to be used in achieving the subtasks. A method, like a task, also assumes a certain model, but usually one more strictly constrained, of input and output *data structures* and semantic constraints on them. The semantic constraints of the method are usually therefore a superset of the constraints of several tasks, potentially indexed by all the latter. The method will also specify explicitly or implicitly a *control structure* that defines the order in which the subtasks will be accomplished, a *dataflow structure* that specifies data connections between subtasks, and possibly the *domain-specific knowledge* (a special input structure) required for the successful application of this method (e.g., certain domain-specific tables and an appropriate look-up procedure). Methods that are not decomposed into subtasks are called *mechanisms*. Some of these mechanisms, used by the problem-identification subtask of the skeletal-plan–refinement method, are the temporal-abstraction mechanisms (Figure 1).

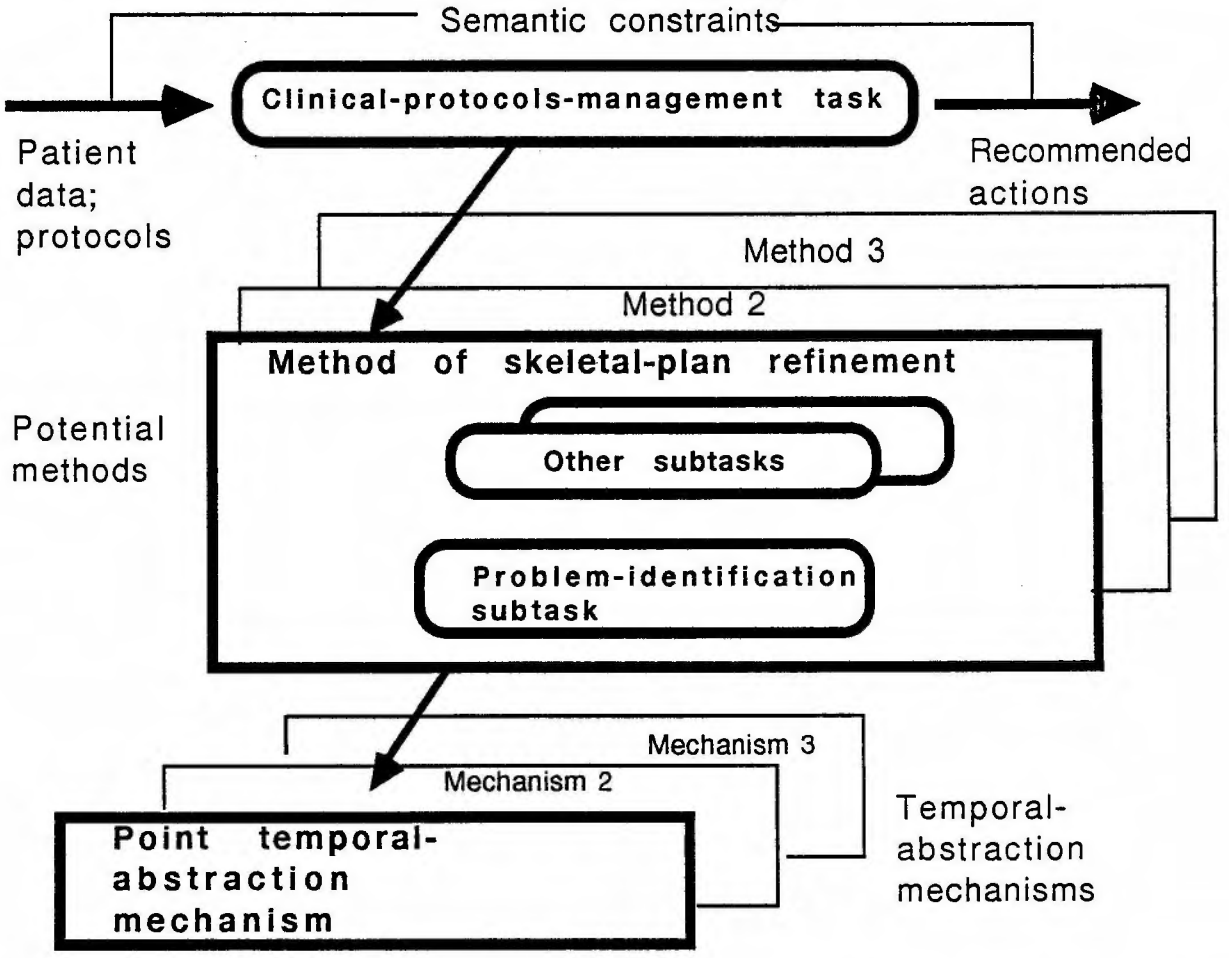


Figure 1: Incremental decomposition of the task of clinical-protocol management by the method of skeletal-plan refinement. The task is defined by the input and output data structures and by the semantic constraints imposed on them. The protocol-management task is decomposed into several subtasks. One of the subtasks is problem identification, which uses mainly temporal-abstraction mechanisms. See the text for further details regarding a method's structure.

The knowledge engineer building a knowledge-based system will select from the method library the closest-fitting method for her task, or will configure the system from scratch using existing methods and mechanisms. In either case, she will instantiate the methods to be used for the subtasks, and thus will recursively specialize the original method. She also will specialize the input and output data structures used by the tasks, thus creating domain-specific labels for every concept. The output of PROTÉGÉ–II will therefore contain both a specific problem solver configured by the knowledge engineer and a custom-tailored knowledge-acquisition tool that can acquire task-specific knowledge needed to solve the task, using the concepts and methods defined by the PROTÉGÉ–II user (Puerta et al., 91). The general temporal-abstraction mechanisms will be specialized at the phase of defining the task model by the PROTÉGÉ–II user, who will fill in some of the missing knowledge. Additional domain- and task-specific knowledge, needed to instantiate the temporal-abstraction mechanisms fully, will be added by the expert physician working with the knowledge-acquisition tool that PROTÉGÉ–II will generate. As in PROTÉGÉ, the end result will be a custom-tailored expert system, designed both to solve the task defined by the PROTÉGÉ–II user and to reason about temporal abstractions in that domain and task.

3. THE TEMPORAL-ABSTRACTION MECHANISMS

We have identified several basic *temporal-abstraction* mechanisms. These mechanisms are general and can be used in most domains and tasks involving management of time-oriented data. The temporal model we shall assume for this discussion includes both time *intervals* and time-stamped *points*. Points are zero-length intervals. An interval I is represented by an ordered pair of time points standing for its end points: [I.start, I.end]. An external *event* (an action or process) is represented by an interval to which the appropriate event is attached. This model is essentially the point-based temporal model as explicitly stated by Shoham (Shoham, 1987; Shoham & Goyal, 1988), although the representation of temporal events (which serve as contexts) as intervals, and the actual syntax we use, resembles the formalism proposed by Allen (1984). Both intervals and points have parameter names and parameter values associated to them. Parameters can be *primitive*, attached only to time points, and have raw numeric values (e.g., "WBC_COUNT is 2000"), or *abstract*, attached to either time points or intervals, and have one of certain predefined values (e.g., "WBC_COUNT_STATE is HIGH"). We shall denote the value of parameter π in the beginning of interval I_1 as I_1.start.π (see Figure 1). In general, we are using a *reified* representation (Shoham and Goyal, 1988), in which the proposition "the WBC count is decreasing during interval I_1" is stated formally as True(I_1, DECREASING(WBC)), thus separating the temporal component of the sentence from the atemporal component. For simplicity, however, we shall represent this proposition as DECREASING(I_1, WBC). If an interval has any abstraction attached to it, such as "HIGH WBC count," we shall refer to it as an *abstracted interval.* Using this model, we can describe several temporal-abstraction mechanisms (Table 1), which can be used explicitly, once they and their implied knowledge-acquisition requirements are defined. In general, we distinguish between three abstraction types, depending on the abstraction type of the abstracted point or interval: *state*, *gradient*, and *rate* abstractions. These correspond, respectively, to the classification of the magnitude of the parameter, the sign of the derivative of the parameter during the interval, and the magnitude of the derivative of the parameter during the interval (e.g., LOW, DECREASING, and FAST abstractions for the WBC parameter). The state abstraction is always possible, even with qualitative parameters having only a nominal scale (e.g., color); the gradient and rate abstractions are only meaningful for parameters which have at least an ordinal scale (e.g., ordinal numbers) or an interval scale (e.g., temperature), respectively. There might turn out to be other important types (e.g., for parameters that have a ratio scale with a zero point, like weight), but the state, gradient and rate abstraction types seem to cover most of the numerical and qualitative data in our domains.

Figure 2 shows an example of temporal points and intervals and their associated parameters. Table 1 provides a summary of the temporal-abstraction mechanisms. We shall use the example of abstracting a LOW(WBC) episode by using several abstraction levels from numeric WBC count data. Note that the temporal-abstraction mechanisms do not operate in any fixed order, but rather iterate alternately, driven by the currently available data and previously derived abstractions (Figure 2).

Mechanism	Input	Output	Required knowledge	Purpose
point temporal abstraction	one point and associated parameters	abstracted point	look-up tables, parameter distributions	classify, transform
temporal inference	one or more abstracted intervals	abstracted sub- or super-interval	parameter- and abstraction-specific inference properties	perform sound logical inference
temporal interpolation	two points, or two abstracted intervals	abstracted interval	maximal gap function and all its arguments, parameter distribution	join points and intervals

Table 1: The temporal-abstraction mechanisms

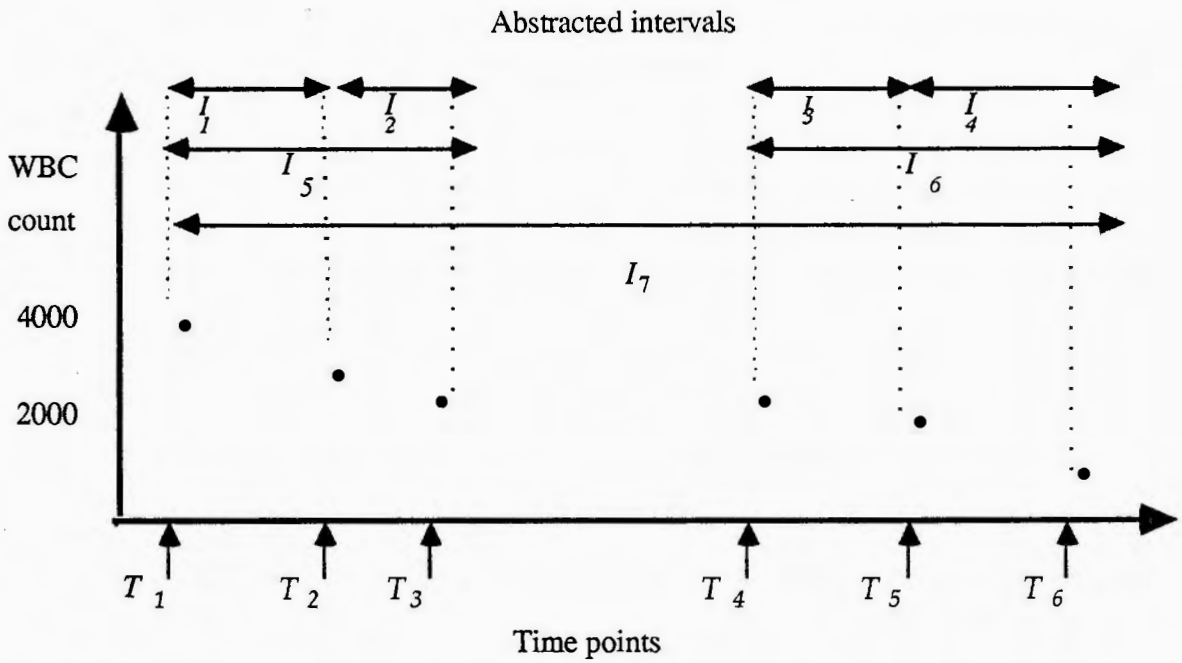


Figure 2: Time points and intervals and their associated parameter values. The parameter is white-blood-cell (WBC) count. Time points T_1 and T_2 are abstracted by point temporal abstraction into a LOW(WBC) classification. They are then joined into a LOW(WBC) interval I_1 by temporal interpolation over the gap $[T_1, T_2]$. Intervals I_1 and I_2 are abstracted by temporal inference into a LOW(WBC) interval I_5, as are intervals I_3 and I_4 into interval I_6. Intervals I_5 and I_6 can then be joined into a longer LOW(WBC) interval I_7 by another application of temporal interpolation, interpolating over the gap interval $[T_3, T_4]$. The DECREASING(WBC) abstraction for the same intervals would be computed similarly.

3.1 Point Temporal Abstraction

The point temporal-abstraction mechanism accepts a time point and its associated parameters as input, and returns an abstracted state as output. It is used for two related purposes. The first is *classification* of an input time point by the latter's parameters; examples include HIGH, MEDIUM, or LOW WBC count values, or any finite, domain-specific number of categories. The knowledge-acquisition requirements include a state list, and range tables that map primitive- or abstract-parameter values into discrete-state values. The classification might be sensitive to patient parameters (e.g., age) and to the relevant context (e.g., a chemotherapy-protocol event). The second purpose, *computational transformation,* is really an extension of classification. It uses a function that maps one or more parameters into another, possibly abstract, parameter. For example, the transformation function computes the percentage of white blood cells that are lymphocytes at a specific time point and maps this value into a predefined category. This category might depend on the context. Thus, computational transformation might use additional parameters, might involve additional computation, and might require special function tables that are specific for the domain, the task and the context. In both cases, we distinguish between a *single-parameter* point abstraction, that maps a single (usually primitive) parameter directly into its state-abstraction values, and a *multiple-parameter* point abstraction, that maps several parameters into one abstract value. As will be noted later, this has some significance regarding the definition of changes in an abstract parameter across time intervals. For instance, an abstract parameter might be defined as INCREASING if all the parameters.taking part in the abstraction are qualitatively proportional to it, as defined by Forbus (1985) and are INCREASING themselves, though the value of the state abstraction of the parameter has not changed.

Other approaches to the problem of mapping several parameter values into discrete categories, and the implications that these approaches would have for the knowledge-acquisition requirements, are discussed at the end of this paper. In either case, a state abstraction is created at the time point in consideration.

3.2 Temporal Inference

The temporal-inference mechanism involves logically sound inference regarding temporal intervals. It uses one or more temporal intervals to characterize a subinterval or a newly defined superinterval. For instance, certain abstractions, when known for interval I_1, can be inferred for every subinterval I_2 that is contained in I_1 (e.g, the characterization "patient in coma"). These abstractions have the property sometimes referred to as the *interval-downward–hereditary* inferential property (Shoham 1987). This property is not valid, for instance, in the case of the NONMONOTONIC gradient abstraction for any parameter. Joining two touching intervals, previously classified as DECREASING for some parameter, into one DECREASING superinterval, uses another inferential property—namely, the *claylike* property (Shoham and Goyal, 1988) or *concatenable* property (Shoham, 1987)—of the decreasing abstraction for any parameter (see Figure 2). Both the downward-hereditary property and the concatenable property can be assumed valid as default inferential properties for most abstractions, unless specified otherwise. Other inferential properties we defined, such as *open-upward-hereditary* (the abstraction can be inferred for any superinterval of the abstracted interval), are assumed invalid as a default. Certain parameters, however, are constants and can be inferred for any interval, once a value is set for some interval (e.g., gender). In addition, certain abstraction instances, such as the gradient abstraction NONMONOTONIC, are open-upward-hereditary for most domains, regardless of the parameter involved. Thus, these inferential properties should exist as defaults, to be overriden for any specific parameter and abstraction by the knowledge engineer defining the basic concepts and entities of the task or by the domain expert elaborating the temporal-abstraction knowledge. The defaults and their modifications can be summarized as an *inference-properties table* where every

tuple (π, $\emptyset$,ρ, τ) represents the fact that the inferential property ρ, for abstraction $\emptyset$ of parameter π, has the truth value τ ($\tau \in$ {TRUE, FALSE}).

Combining two touching intervals—one in which the parameter is abstracted as INCREASING and one in which the parameter is abstracted as DECREASING—into a superinterval abstracted as NONMONOTONIC, is another type of sound temporal inference. Another option in this case would be to denote the resulting interval as DECREASING–INCREASING for that parameter, possibly preserving the lengths of the particular subintervals involved (we shall elaborate more on this option in the discussion section). In general, a complete *abstraction-inference table* should be constructed for all the task-relevant abstraction combinations. For instance, in the case of the gradient abstraction we have defined a default, domain-independent qualitative $\oplus$ operation for joining gradient-abstracted intervals, where the allowed symbols are {DECREASING, INCREASING, SAME, NONINCREASING, NONDECREASING, NONMONOTONIC}. The abstractions SAME, DECREASING, and INCREASING can be the results of primary interpolation over two points using standard algebra; the rest are concluded only as the result of secondary mechanisms (e.g., DECREASING $\oplus$ SAME = NONINCREASING). This is a modification of the $\oplus$ operation in the Q1 algebra (Williams, 1988), where the sign of the derivative of the parameter, whose gradient is abstracted, is mapped in the following fashion: {0} $\rightarrow$ SAME, {+} $\rightarrow$ INCREASING, {-} $\rightarrow$ INCREASING, {?} $\rightarrow$ NONMONOTONIC. We have added the NONINCREASING symbol for the set {0, -} and the NONDECREASING symbol for the set {0, +}. It can be shown that this extension still preserves associativity [$(X \oplus Y) \oplus Z = X \oplus (Y \oplus Z)$] and commutativity [$X \oplus Y = Y \oplus X$]. As usual, the problem of qualitative algebras remains: there is no additive inverse. Table 2 presents a summary of the inference table for the secondary gradient-abstraction operation.

Most of the entries in the abstraction-inference table can exist as domain-independent defaults, or can be filled in ahead of time by the knowledge engineer, but the domain expert can add more abstractions and can then define the result of concatenating meeting intervals already abstracted by temporal abstractions. As we shall see, the abstraction-inference table is also used by the temporal-interpolation mechanism. The abstractions concatenated, however, should be of the same type—for instance, a state abstraction-type (e.g., HIGH or LOW) or a gradient abstraction-type (e.g., INCREASING or SAME).

$\oplus$	DEC	NONINC	SAME	NONDEC	INC	NONMON
DEC	DEC	NONINC	NONINC	NONMON	NONMON	NONMON
NONINC	NONINC	NONINC	NONINC	NONMON	NONMON	NONMON
SAME	NONINC	NONINC	SAME	NONDEC	NONDEC	NONMON
NONDEC	NONMON	NONMON	NONDEC	NONDEC	NONDEC	NONMON
INC	NONMON	NONMON	NONDEC	NONDEC	INC	NONMON
NONMON	NONMON	NONMON	NONMON	NONMON	NONMON	NONMON

Table 2: The $\oplus$ operation for the gradient temporal-abstraction inference. The rows denote the value of the first abstracted interval; the columns denote the value of the second abstracted interval. The entries represent the abstracted values of the concatenated interval. DEC = DECREASING; INC = INCREASING; NONMON = NONMONOTONIC; NONINC = NONINCREASING; NONDEC = NONDECREASING.

3.3 Temporal Interpolation

The temporal-interpolation mechanism accepts two points or two intervals with their associated primitive or abstract parameters, and returns an abstracted superinterval as output, interpolating over the gap between them. For instance, we need temporal interpolation when classifying an interval as DECREASING when the start point parameter value in that interval is greater then the end point parameter value (and the parameter values of the intermediate time points are unknown), or when abstracting two sufficiently close intervals, both abstracted as episodes of LOW(WBC counts), into a longer one (see Figure 2). We can distinguish between two types of temporal interpolation, depending on the temporal component of the input. *Primary interpolation* accepts two points and their parameters and returns an abstracted interval. *Secondary interpolation* accepts two abstracted intervals, and returns an abstracted superinterval.

Primary state interpolation accepts two points T_1, T_2 and their associated abstract parameters (e.g., a state abstraction with the value LOW for the same parameter), and bridges the gap between them to attach the abstraction to the interval $[T_1, T_2]$. *Primary gradient interpolation* accepts two points T_1, T_2 and their associated values of the same primitive (or possibly, abstract) parameter π and, if certain conditions hold, returns the interval $[T_1, T_2]$ classified as DECREASING, INCREASING, or SAME with respect to the change in π. *Primary rate interpolation* classifies change rates in π (e.g., STABLE, SLOW, or FAST) between two points that have associated values of the same primitive (or possibly, abstract) parameter. *Secondary state interpolation* occurs when two intervals I_1 and I_2 are joined into a new interval $Ij = [I_1.\text{start}, I_2.\text{end}]$ over which a state ϕ is true, given that state ϕ is true in both joined intervals. An example is joining two temporally close intervals over which the WBC count was abstracted as LOW. *Secondary gradient interpolation* joins two intervals, over which the parameter change was previously abstracted, into an INCREASING, DECREASING, SAME, NONDECREASING, NONINCREASING, or NONMONOTONIC abstracted superinterval. *Secondary rate interpolation* joins two intervals previously abstracted with respect to rate. Note that a parameter, such as the WBC count, might be abstracted as INCREASING in a certain interval with respect to the gradient abstraction, and be abstracted as NORMAL with respect to the state abstraction type.

Temporal interpolation requires that the difference between the two points or intervals be less than a certain time *gap*. Within that time gap the behavior of the parameter can then be assumed stable with respect to the abstraction ϕ (e.g., LOW or INCREASING) and can be abstracted by it. The maximal allowed gap must be a domain-, task-, and context-dependent function (e.g., the maximal allowed gap for LOW(WBC counts) in the domain of oncology, the task of caring for patients using protocols, and the context of patients receiving X-ray therapy). The arguments of the function for each specified context include the parameter we are abstracting and the specific abstraction we have in mind. They also include a measure of the rate of change of the parameter before and after the time gap. As a first approximation to the rate of change, we use the length of the intervals before and after the gap. Thus, for every context ξ, we denote the maximal-gap function Δ as $\Delta(\pi,\phi,\text{L}(I_1),\text{L}(I_2),\xi)$ of the specific abstracted parameter π, the abstraction value (implicitly including the type) ϕ, and the lengths $\text{L}(I_1)$, $\text{L}(I_2)$ of the intervals to be joined in the context ξ. The context is usually given by the co-occurring events. For instance, in any context, joining two intervals where the WBC count state was classified as LOW, into a longer interval whose WBC count state is classified as LOW, depends on the time gap separating the two intervals, on the properties of the WBC count state abstraction for the value LOW, and on the length of time in which the LOW property was known both before and after the time gap (see Figure 2). In general, if intervals I_1 and I_2 are sufficiently close, as judged by the relevant Δ gap function, for a certain parameter π and state ϕ, we say that $I_1 \approx I_2$ (or I_2 *extends* I_1) with respect to π, ϕ. Note that interpolation always starts by joining two time points (zero-length intervals) T_1 and T_2, in which ϕ

is true for π, and where $T_1 \approx T_2$ with respect to joining over π, ϕ, into a new interval $[T_1, T_2]$, over which ϕ is true for π. That is, primary interpolation requires that at least $L([T_1, T_2]) \leq \Delta(\pi,\phi,0, 0, \xi)$. We shall usually omit the context argument of the Δ function, since the following discussion holds for any context.

In general, state, gradient, or rate interpolation requires additional conditions, besides an upper bound on the gap between the intervals, to preserve consistency. These conditions, as well as the maximal gap specified by the gap function, are summarized by an extension of the abstraction-inference table, an *abstraction-interpolation table*, defining the interpolation operation for every abstract parameter (e.g., WBC_LEVEL) and combination of abstraction values. (e.g., INCREASING and SAME). For example, we need to check that, when we use secondary temporal interpolation to join two intervals I_1, I_2, both abstracted as DECREASING for π, into a superinterval I_j that is abstracted as DECREASING for π, the value of π has indeed decreased, or at least has not increased above a certain predefined threshold during the time gap [I_1.end, I_2 .start] (see Figure 2). In other words, we have to check that $I_1.\text{end}.\pi \geq I_2.\text{start}.\pi - C_\pi$, where C_π represents a measurement variation for π—the maximal increment in parameter π, below which a change in π will not be considered as an increase. C_π is a function of π, $f_C(\pi)$ that is defined either by the domain expert or through analysis of the distribution of π. Using the C_π parameter, we can ignore minor absolute changes in the value of π that are less than a certain threshold when we wish to identify general qualitative trends. On the other hand, in the case of primary temporal interpolation for the INCREASING gradient abstraction, for instance, we require that $T_2.\pi - T_1.\pi \geq C_\pi$. Table 3 provides an example of the secondary temporal-interpolation operation for the DECREASING abstraction.

In the case of gradient and rate abstractions for abstract parameters whose state values are concluded by a multiple-parameter point temporal abstraction, it should first be noted whether there exists more than a nominal scale for the possible output states (else, there is no meaning to the gradient and rate abstractions). Even if the resulting states can be ranked on an ordinal (for gradient abstractions) or on an interval (for rate abstractions) scale, there are in general two options as to what is the meaning of the gradient- or rate-interpolation operation

Interval I_1 abstraction for parameter π	**Interval I_2 abstraction for parameter π**	**Additional required conditions for the interpolation operation**	**Temporal interpolation: I_j=[I_1.start,I_2.end]**
DEC(I_1,π)	DEC(I_2, π)	1) $I_1.\text{end}.\pi \geq I_2.\text{start}.\pi - C_\pi$ 2) $I_2.\text{start} - I_1.\text{end} \leq \Delta(\pi, \text{DEC}, L(I_1), L(I_2))$	DEC(I_j, π)
DEC(I_1, π)	INC(I_2, π)	———	NONMON(I_j, π)
DEC(I_1, π)	NONDEC(I_2, π)	———	NONMON(I_j, π)
DEC(I_1,π)	NONINC(I_2, π)	1) $I_1.\text{end}.\pi \geq I_2.\text{start}.\pi - C_\pi$ 2) $I_2.\text{start} - I_1.\text{end} \leq \Delta(\pi, \text{NONINC}, L(I_1), L(I_2))$	NONINC(I_j, π)
DEC(I_1, π)	SAME(I_2, π)	1) $I_1.\text{end}.\pi \geq I_2.\text{start}.\pi - C_\pi$ 2) $I_2.\text{start} - I_1.\text{end} \leq \Delta(\pi, \text{NONINC}, L(I_1), L(I_2))$	NONINC(I_j, π)
DEC(I_1, π)	NONMON	———	NONMON(I_j, π)

Table 3: The temporal-interpolation operation for one parameter, two Intervals, and the DECREASING gradient abstraction; Δ is the gap function for the given context; C_π is the threshold increment defined for parameter π.

The *direct* option is to consider the state values of the abstract parameter as its only values, assuming they are ranked on the ordinal scale from lowest to highest (e.g., LOW, MODERATE, HIGH, EXTREME). Then we can define that an interval spanning the distance between two multiple-parameter state abstractions, can have attached to it also, for instance, an INCREASING gradient abstraction, if and only if a positive change in the rank of the states of the two end points of the abstracted interval has taken place. An equivalent definition can apply to rate abstractions. The *indirect* option is to define the meaning of an INCREASING promary interpolation for a single- or multiple-parameter abstraction such that if all the qualitatively proportional parameters determining π are INCREASING, then π can be abstracted as INCREASING, even if the value of its state over the interval I has not changed. To denote the direction of the qualitative relationship we use the notation $\pi_i \propto_{Q+} \pi_j$ or $\pi_i \propto_{Q-} \pi_j$. This notation is taken from Forbus (1985) and means, in the case of $\pi_i \propto_{Q+} \pi_j$, that π_i is qualitatively proportional to π_j (i.e., there is a function that determines π_i and is monotonically increasing in its dependence on π_j, and determines π_i). In the case of $\pi_i \propto_{Q-} \pi_j$ the dependence is monotonically decreasing. Thus, the INCREASING abstraction would be defined in the following recursive fashion:

1. A primitive parameter π is INCREASING over interval I if all the conditions for primary interpolation of π over I (i.e., a check for the time gap vs. the $\Delta(\pi, \text{INC}, 0, 0)$ function, and a check for the parameter-value change direction and change magnitude of π vs. C_π) hold.

2. An abstract parameterπ is INCREASING over interval I if the rank of the state abstraction of π has increased between I.start and I.end (i.e., $I.\text{start}.\pi < I.\text{end}.\pi$, and the other conditions for primary interpolation of π hold).

3. Otherwise, an abstract [state] parameter π is INCREASING over interval I if the value of the state of π in the begining of I is the same as the value of the state of π in the end of I ($I.\text{start}.\pi = I.\text{end}.\pi$), and all the abstract or primitive parameters π_i from which the state abstraction of π is mapped, such that $\pi \propto_{Q+} \pi_i$, can be abstracted as INCREASING, and all the abstract or primitive parameters π_j from which the state abstraction of π is mapped, such that $\pi \propto_{Q-} \pi_j$, can be abstracted as DECREASING, and the other conditions for primary interpolation of π hold.

The symetric conditions for the DECREASING abstraction are similarly defined, as well as the extension for secondary gradient and rate interpolations between abstracted intervals. If the gradient abstraction is not DECREASING or INCREASING and the other usual conditions hold (e.g. Δ function check) then its value is SAME.

In many domains and tasks (e.g., monitoring intensive-care patients) it is plausible for either the knowledge engineer working at the PROTEGE–II system-definition level, or the domain expert working at the knowledge-acquisition–tool level, to define gradient and rate abstractions for multiple-parameter–determined abstractions using the indirect option. For instance, even in the simple case of a single-parameter abstraction from the WBC count (see Figure 2), a domain expert (a hematologist, in this case) is interested in the fact that the WBC count is not only LOW throughout interval I_7, but is also DECREASING (judging by the primitive parameter it is abstracted from, WBC count). That would still be the case if the WBC count abstraction was a function of several parameters, such as the various types of the white blood cells. These, therefore, are the default semantics we chose for the gradient and rate abstractions in the case of abstract parameters

mapped by multiple-parameter point abstractions. However, that can be overriden by the domain expert.

In general, when π can be abstracted as DECREASING over the gap [I_1.end, I_2.start], we can describe the secondary interpolation more formally: We apply *primary temporal interpolation* to the DECREASING abstraction of π over the *gap interval* between the two intervals. That is, we infer by interpolation DECREASING([I_1.end, I_2.start], π). At this point, the *temporal inference* mechanism suffices to infer DECREASING(I_j, π) by iterative joining of I_1, I_j, and I_2. However, in general, the temporal-interpolation mechanism can be seen as an extension of the temporal-inference mechanism in the case of the specific task of joining intervals that have the same abstraction type (assuming the abstraction has the concatenable property), and its conclusions override the somewhat weaker conclusions of the temporal-inference mechanism .

The knowledge-acquisition requirements for the temporal-interpolation operation (see Table 1) include definition of the time units used in the domain, including relative times (e.g., most recent treatment cycle). We also need to acquire a definition of what is a significant change—increment or decrement—for the parameter π, as well as a classification of the various change rates of π in terms of change per unit time. The main knowledge-acquisition requirement, however, is defining the gap function Δ, including additional domain- and task-specific arguments. It is also necessary to define *which* parameters should be tracked by the system (e.g., age always increases and should not be abstracted).

In summary, temporal interpolation requires acquisition of knowledge that will enable us to classify the parameter's state (e.g., HIGH, LOW), relative change (e.g., INCREASING, DECREASING, or NONMONOTONIC) and rate of change (e.g., SLOW, FAST) over an interval, as well as to join two intervals (if possible, given the interpolation table) with any combination of states, gradients (change directions), or change rates of the parameter over the two intervals. This knowledge includes abstraction-specific gap functions that should be tailored to the domain and to the task.

3.4 The Nonmonotonicity of Temporal Abstractions

The three temporal-abstraction mechanisms discussed create state-abstracted points (in the case of point temporal abstraction) and state-, gradient- or rate-abstracted intervals (in the case of temporal inference and temporal interpolation). Unlike actual data attached to time-stamped data points, however, these new, abstracted, data are potentially refutable by any modification or addition to the known data points or events. Since intervals are created by the system only on the basis of repeated domain-independent as well as domain-specific heuristic inference and interpolation, every interval created by the system depends on defeasible assumptions and is potentially retractable.

Thus, one of the inherent requirements of temporal abstractions is the necessity to allow for a defeasible logic. Since data might arrive in the present time, but pertain to an earlier time point, it is necessary to allow for at least a partial revision of the current abstracted evaluation of the history of the time-oriented database. In particular, in domains such as ours (i.e., when progress is being monitored episodically over a long time), we should be able to revise efficiently our former assessment of the situation up to date. For instance, Fagan's VM system (1980), whose domain was the fast-changing intensive-care unit, dealt with incoming data incrementally, but did not update old assessments when newly available data about the past arrived. The need to update

former abstractions when old (but formerly unavailable) data arrive was noted by Long and Russ (Long & Russ, 1983; Russ, 1986). Another phenomenon, consisting of updating former conclusions and revising assessments of former decisions given new data (in particular, the present result of past decisions) is referred to as *hindsight* by Russ (1989). In the first case, we need to evaluate precisely that part of our interpretation and abstraction of the former data that is affected by the newly added old (with respect to the time stamp) data, such as laboratory reports that are returned a few days after the initial assessment was done. In the second case, we need to reflect back our current understanding of the situation to the interpretation of the former events, even if no direct additional information about the past has been uncovered; usually, that understanding is due to current developments, such as the effect of the therapy.

Note that the decisions themselves (i.e., the *events*, in our notation) cannot be changed. Only our interpretation of the data (i.e., the *abstracted intervals*) can change, as well as the assessment of the best decision that *could have been* made in the past, or that can be made in the present. The possible changes include modifying the value of an existing data point, adding a newly known data point to the past history, adding a new, present-time data point, adding (starting) a new event, and modifying knowledge about past events, which constitute the main part of the *context* in which abstractions were carried out. It is also possible, in theory, to add abstracted intervals that do not depend on any abstraction mechanism and thus do not need support from any data points (e.g., "the patient is known to have had AIDS during the interval spanning the past 20 months"). In each case, the newly added information should reflect on the past and present by retracting previous abstractions or adding new abstracted points and intervals.

Note also that while most conclusions depend on the existence of certain facts (e.g., the interpolation between two time point—creating an abstracted interval—depends on the data attached to the original end points), some conclusions depend also on the *nonexistence* of certain facts (e.g., no point of a different abstracted state of the same parameter existing between the two interpolated points). Some of the underlying conditions—in particular, the negative ones—are not trivial to track, and depend implicitly on specific domain knowledge in order to establish new conclusions, or retract correctly old ones, when new or modified data become available.

In our mainly point-based model, changes in the abstracted points and intervals occur incrementally, following updates in either present or past time-stamped data, starting always with a specific point (or points) and propagating the change, as in a truth-maintenance system (TMS), using the abstraction knowledge acquired for the relevant parameters considered. In particular, the updating process uses the known inferential properties and various delta functions to retract conclusions that are no longer valid, or to add new ones. For instance, modifying the WBC value of time point T_2 in Figure 2 to a value that can be abstracted as HIGH not only changes the abstraction attached to the time point T_2, but also causes retraction of the LOW(WBC) abstracted intervals I_1, I_2 (that were created through interpolation depending on T_2), retraction of interval I_5 (that was created by inference depending on I_1, I_2), and therefore (due to the delta fuction of the LOW(WBC) state abstraction), also interval I_7, which was created by interpolation depending on interval I_5. Similary, the DECREASING(WBC) abstractions attached to intervals spanning the same time lengths are retracted. If there were other time points within one of the retracted intervals that inherited one of their abstracted properties from one of the retracted intervals through an inference mechanism, they would have lost that particular abstraction. On the other hand, new abstracted intervals should be created, such as an INCREASING(WBC) abstracted-interval I_1. Other updates would cause changes in a similar fashion.

4. THE IMPLEMENTATION

The temporal-abstraction mechanisms have been implemented in a prototype system, RÉSUMÉ. RÉSUMÉ is written in CLIPS 5.0 (an expert-system shell developed by the NASA Software Technology Branch). CLIPS is written in the C language and allows addition of C functions (as well as other procedural languages), but has an independent production-rule language, a general functional Lisp-like programming language, and the CLIPS Object-Oriented Language (COOL). CLIPS

The knowledge regarding parameter and abstraction hierarchies is stored in the object-oriented part of the RÉSUMÉ system, thus enabling inheritance of default inferential properties, tables, and methods from generic abstraction classes to more specific subclasses of abstractions and events, as well as easy modification to include new knowledge acquired for a particular set of abstractions. The domain-independent inference mechanisms are implemented mainly through combined use of rules and methods; the control is essentially a data-driven one, through the rules, accessing a database of time-stamped data points and abstracted intervals. The temporal-database functions are intended to be a part of the temporal-database system on top of which operates the temporal-abstraction system.

The dependency-tracking mechanisms use the CLIPS TMS capabilities for creating dependencies between data points and intervals, between abstracted points and intervals and between abstracted intervals and other abstracted intervals. Additional extensions to the TMS capabilities, utilizing the abstractions' inferential properties, are used during point and interval updates.

Our initial assessment, limited to examining the clinical management tasks in our domains, suggests that for any particular context in one of our domains, a significant (but tenable) amount of knowledge should be acquired, either through the knowlede engineer during the PROTÉGÉ–II sytem-modeling stage, or from a domain expert, using an automatically-generated KA tool, in order to instantiate the temporal-abstraction mechanisms. These items are mainly a significant change indication for every primitive parameter, the list of potential state and rate abstractions for each primitive and abstract parameter that is relevant to the task, and the maximal-gap Δ functions (currently represented as tables of maximal interval lengths) for the three abstraction types (state, gradient, and rate), for each relevant parameter. Inferential properties and qualitative inference and interpolation tables are mostly context independent, and default values and methods can be inherited through the appropriate abstraction class or can be acquired at only the general domain level.

Further evaluation is needed to assess the usefulness of the symbolic temporal-abstraction mechanisms in various tasks and domains, and to determine the best internal representation, as well as the best interfaces for the knowledge engineer and the domain expert. These interfaces are needed for acquiring the various inference tables, and, in particular, the Δ functions.

5. DISCUSSION

There are distinct advantages to using two-stage knowledge-acquisition tools (Musen, 1989). This division separates the task that can be best performed by a knowledge engineer from the task that can be performed easily by a domain expert, once the problem-solving method and the domain-specific concepts have been defined. The same advantages apply to acquisition of temporal-abstraction knowledge. The knowledge engineer who uses PROTÉGÉ–II to configure a new

knowledge-based system and to define the task model can also define the underlying temporal model and the semantics of the necessary temporal-abstraction mechanisms. Additional task-specific knowledge needed to instantiate the temporal-abstraction mechanisms (e.g., interpolation tables) can be added by the application expert using the knowledge-acquisition tool generated by PROTÉGÉ–II. The expert also will be able to use the temporal-abstraction mechanisms to describe efficiently certain protocol procedures. In the medical domain, such a procedure might determine whether the patient has had a drug-toxicity episode. Thus, there are two knowledge-acquisition levels.

In general, the knowledge-acquisition process in PROTÉGÉ–II is performed top down, a task being decomposed recursively into its methods and mechanisms. However, the temporal mechanisms described in this paper introduce interesting design considerations. The temporal-abstraction mechanisms are a prerequisite for many planning tasks—particularly for those in the domain of clinical management. Thus, it may be more convenient for knowledge engineers working in temporally oriented domain to instantiate the temporal mechanisms fully, by adding the domain- and task-specific knowledge, regardless of the top-level problem-solving method chosen.

In addition to the three basic temporal-abstraction mechanisms described, we have noticed the existence of higher-level temporal-abstraction mechanisms that are required for abstracting certain data patterns over time, such as abstracting an episode of drug toxicity from a state of a LOW WBC count lasting more than 2 weeks in a patient receiving certain drugs. The higher-level mechanisms extend temporal inference and temporal interpolation by abstracting over multiple intervals and parameters, as well as by using heuristic domain- and task-specific knowledge. Higher-level abstractions can use task-specific time units (e.g., chemotherapy cycles or relative, nonabsolute time references, such as "the first toxic episode"); they are intricate and require further research on a more uniform representation. Initial work indicates that such mechanisms can use a language that depends on the existence of the three basic abstraction mechanisms and their respective outputs.

We are considering the inclusion of more information in the qualitative inference tables and rules, as mentioned in our discussion of the temporal-inference mechanism. An INCREASING state followed by a DECREASING state for the WBC-count parameter might be summarized concisely as "NONMONOTONIC WBC count gradients during the past 3 weeks," but might also be summarized as "INCREASING–DECREASING(WBC)," and even as "2 weeks of INCREASING WBC counts followed by 1 week of DECREASING WBC counts." The last option requires a language where we can use expressions such as "if there are two episodes, in a certain context, of INCREASING(WBC) for less than 2 weeks followed by DECREASING(WBC) for at least 3 weeks, then infer the hematologic state abstraction BONE-MARROW-SUPPRESSION-GRADE-I." Such *temporal templates* can be acquired from application experts. An existing temporal pattern can then be compared against a temporal template, to decide how closely the pattern matches the template. This matching would constitute a higher-level abstraction mechanism, and would depend on the existence of abstractions formed by the three basic temporal-abstraction mechanisms.

The choice of the amount of information included in an abstraction, as exemplified by the NONMONOTONIC abstraction, demonstrates another design consideration. The output of the temporal abstraction mechanisms depends on the intended use of the abstraction. Different users need different parameters and different levels of abstraction. The reasoning module of a planning system needs detailed abstraction output, including lengths of intervals, to be able to reason about plan-execution problems and about possible modifications to planned or executed events and their respective parameters, which can affect the states on which the system is focusing. In the medical domain, another user might be a dermatologist called for a consultation regarding a patient's skin rash; this expert might need only a brief summary of the parameters relevant to him, over a certain

limited period, and would not want more details than are necessary for his purpose (say, assessment of the patient's current state). High-level abstractions (e.g., "a toxicity episode, typical of the drug vincristin, lasting during the past 3 weeks") might be sufficient. In addition, each user has different levels of knowledge about the domain and task. For instance, in the clinical domain, the expert dermatologist might fill in missing details regarding a drug-allergy event himself, but the internist would need further elaboration. We consider this user model (of the different parameters and abstraction levels required) to be part of the system's knowledge. The knowledge needed to instantiate the user model might have to be acquired at three different levels. The three levels correspond to the three user types: the knowledge engineer, working with the metalevel PROTÉGÉ–II tool; the domain expert, working with the output of PROTÉGÉ–II, an OPAL-like knowledge-acquisition tool; and the actual user, working with the final knowledge-based application system produced by a combination of the problem-solving method chosen by the knowledge engineer with the knowledge base instantiated by the domain expert using the knowledge-acquisition tool.

We wish to acquire knowledge not only for identifying problems and generating plans, but also for revising an existing plan and, in particular, for modifying it during execution. We are defining a set of general execution-modification mechanisms, to be used during the plan-execution time, suitable for plans that can be represented as a set of temporal events (e.g., administration of a drug) and states (e.g., fever). These states and events correspond to Simmons' objects and events, which underlie his generic strategies for debugging plans during planning time (1988). The execution-modification mechanisms examine the abstracted states attached to a temporal interval and the events that might have caused or might modify each state. Thus, revising plans during execution time requires the ability to abstract temporal points and intervals and to understand the events affecting different abstractions in the domain and task.

Classifying time-stamped numerical data into discrete categories using predefined ranges of values, as is done by the point temporal-abstraction mechanism, presents several subtle problems, one of which is the *threshold* problem. The expert abstracting a body temperature below 98° F as a LOW temperature does not necessarily mean that 98.01° F should be considered as HIGH. Such use of the definition might lead to erroneous conclusions when similar cases are compared, where the temperatures are essentially the same. One way to approach the threshold problem is to represent the cut-off points between different categories as a set of constraints, using heuristics to choose a specific cut-off value when necessary.[1] These constraints must be acquired from the domain expert. Another possibility is to represent cut-off points in a probabalistic fashion, assigning to each value (or range) a certain probability of belonging to the LOW or HIGH abstraction class. In this case, the probability distribution of every abstraction across the parameter values must be acquired. The process of mapping several numerical data parameters into a discrete, predefined category needs further investigation.

Most knowledge-based systems either do not represent temporal data at all, or do not reason about such data explicitly, and in particular do not represent the knowledge-acquisition requirements, including the system's underlying inferential assumptions, which are needed to instantiate the system's temporal abstraction mechanisms in another domain or task. Several qualitative physics systems use temporal abstractions, but usually take the latter for granted and do not create new

---

[1] N. Goyal and Y. Shoham, personal communication, 1991.

ones (Bobrow, 1985). Other workers (Long & Russ, 1983; Russ, 1986; Russ, 1989) have used symbolic abstraction modules that are encoded procedurally. Russ in particular has emphasized the value of separating the time-reasoning mechanisms from the expert-system time-independent rules. Van Beek (1989, 1990) discusses several approaches to the problem of computing efficiently the strongest possible assertions necessarily or possibly true about relations between temporal intervals when certain relations between the intervals have been defined using Allen's interval-based language (1984). Such computations are useful when queries are addressed to the temporal database regarding possible or necessary relations among several events, if the information about these events is known only as a set of partial-order constraints on the start and end points of the event intervals. Systems have been built that can reason in specific clinical areas about time-oriented parameters with the intent of presenting high-level summaries to a physician user (Downs, 1986; de Zegher-Geets, 1987). Usually, both the specific assumptions enabling different types of inferences and the knowledge requirements for the system remained implicit. Our work, however, elucidates such temporal-abstraction mechanisms so that the knowledge-acquisition requirements become apparent.

There are several advantages in our application area to the acquisition of explicit knowledge regarding the basic temporal-abstraction mechanisms. This knowledge is important for the acquisition of the further domain- and task-specific knowledge needed to recognize the patient's past and current state, including the main clinical problems needing treatment, the difficulties currently and previously encountered while executing the treatment plan, and the complications likely to occur. These advantages are common to other domains where data are monitored over time. Separating the temporal-abstraction mechanisms from the planning and execution components of a decision-support system allows the system to reason about the data off-line, using data directly from a temporally oriented database. Abstractions also enable a concise representation of large volumes of data for different types of users. The knowledge required to instantiate the domain-independent temporal-abstraction mechanisms for any particular domain and task should be formulated precisely and should be acquired from the domain expert. The knowledge-acquisition process can use automatically generated knowledge-acquisition tools tailored to the domain and to the task, such as the knowledge-acquisition tools generated by PROTÉGÉ–II. It is our goal to generalize the design process of knowledge-based systems for specific domains and tasks using multiple problem-solving methods, and to include in this process an instantiation of the appropriate temporal-abstraction mechanisms. These mechanisms will facilitate carrying out common tasks such as problem identification, plan generation, and plan revision, and will be instantiated in a domain- and task-specific manner.

ACKNOWLEDGMENTS

This work has been supported in part by grant LM05157 from the National Library of Medicine, by grant HS06330 from the Agency for Health Care Policy and Research, by a grant from the Institute for Biological and Clinical Investigation at Stanford University, and by a gift from Digital Equipment Corporation. Computer support was provided in part by grant RR05453 from the Biomedical Research Support Grant Program of the National Institutes of Health, and by the SUMEX-AIM resource, supported by grant LM05208 from the National Library of Medicine.

We thank John Dawes, John Egar, Angel Puerta, James Winkles, and Joseph Schreiner for their contributions to the design of the PROTÉGÉ–II system; Yoav Shoham, Nita Goyal, and Becky Thomas for illuminating discussions regarding temporal abstractions, and Lyn Dupré for her helpful harsh criticism of the style of earlier versions.

REFERENCES

Allen, J.F. (1984). Towards a general theory of action and time. *Artificial Intelligence* **23**:123–154.

Bobrow, D.G. (ed.)(1985). *Qualitative reasoning about physical systems.* Cambridge, MA: MIT Press.

Chandrasekaran, B. (1990). Design problem solving: a task analysis. *AI Magazine* **11**:59–71, winter.

Downs, S.M., Walker M.G., and Blum, R.B. (1986). Automated summarization of on-line medical records. *Proceedings of the Fifth World Congress on Medical Informatics*, pages 800–804, Washington D.C.

De Zegher-Geets, I.M., Freeman, A.G.,Walker, M.G., and Blum, R.B. (1987). Computer-aided summarization of a time-oriented medical data base. *Knowledge Systems Laboratory Report KSL–87–18*, Department of Computer Science, Stanford University, Stanford, CA.

Fagan, L.M. (1980). VM: representing time-dependent relations in a medical setting. Ph.D. thesis, Department of Computer Science, Stanford University, Stanford, CA.

Forbus, K.D. (1985). Qualitative process theory. In Bobrow, D.G. (ed.) *Qualitative reasoning about physical systems.* Cambridge, MA: MIT Press.

Kahn, M.G. (1991). Extensions to the time-oriented database model to support temporal reasoning in medical expert systems. *Methods of Information in Medicine* **30**:4–14.

Long, W.J., and Russ, T.A. (1983). A control structure for time dependent reasoning. *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 230–232, Karlsruhe, West Germany.

Musen M.A., Fagan, L.M., Coombs, D.M., and Shortliffe, E.H (1987). Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man–Machine Studies* **26**:105–121.

Musen, M.A. (1989). *Automated generation of model-based knowledge-acquisition tools.* San Mateo, CA: Morgan Kaufmann.

Puerta, A.R., Egar, J.W., Tu, S.W., and Musen, M.A. (1991). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Proceedings of the Sixth Workshop on Knowledge Acquisition for Knowledge-Based Systems,* Banff, Alberta, Canada.

Russ, T.A. (1986). A system for using time-dependent data in patient management. *Proceedings of the Fifth World Congress on Medical Informatics*, pages 165–169, Washington D.C.

Russ, T.A. (1989). Using hindsight in medical decision making. *Proceedings of the Symposium on Computer Applications in Medical Care,* pages 38–44, Washington D.C.

Shoham, Y., and Goyal, N. (1988). Temporal reasoning in artificial intelligence. In Shrobe, E.H. (ed.), *Exploring Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Shoham, Y. (1987). Temporal logics in AI: semantical and ontological considerations. *Artificial Intelligence* **33**:189–104.

Shortliffe, E.H. (1986). Medical expert systems—knowledge tools for physicians. *Western Journal of Medicine* **145**:830–839.

Simmons, R.G. (1988). A theory of debugging plans and interpretations. *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94–99, St. Paul, Minnesota.

Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J.C., Shortliffe, E.H., and Fagan, L.M. (1989). Episodic skeletal-plan refinement based on temporal data. *Communications of the ACM* **32**:1439–1455.

Tu, S.W., Shahar, Y., Dawes, J., Winkles, J., Puerta, A.R., and Musen, M.A. (1991). A problem-solving model for episodic skeletal-plan refinement. *Proceedings of the Sixth Workshop on Knowledge Acquisition for Knowledge-Based Systems,* Banff, Alberta, Canada.

Van Beek, P. (1989). Approximation algorithms for temporal reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1291–1296, Detroit, MI.

Van Beek, P. (1990). Reasoning about qualitative temporal information. In *Proceedings of the Eight National Conference on Artificial Intelligence*, pages 728–734, Boston, MA.

Williams, B.C. (1988). MINIMA: A symbolic approach to qualitative algebraic reasoning. *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94–99, St. Paul, MN.

USING KNOWLEDGE ACQUISITION TOOLS TO SUPPORT CREATIVE PROCESSES

Mildred L G Shaw & Brian R Gaines

Knowledge Science Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4
mildred@cpsc.ucalgary.ca & gaines@cpsc.ucalgary.ca

Abstract: Many of the tools developed within the knowledge acquisition research community to support the knowledge engineer in developing knowledge-based systems are based on methodologies from psychology, education or management designed to support people in developing overt conceptual structures. Often the root discipline already has explicit models of the roles of these methodologies in supporting the human creative process. Given the technical progress in developing a variety of interactive knowledge acquisition tools in recent years it is appropriate to investigate the applicability of these tools to their original domains of application. This paper reports some initial experiments on the application of knowledge acquisition tools to support first year graduate students in establishing their own research directions and identities.

INTRODUCTION

Human creativity is a puzzle within an artificial intelligence and knowledge acquisition context. We expect creativity to be associated with intelligence, yet factor analyses of intelligence tests and divergent thinking tests suggest that creativity is a distinct factor of ability independent of intelligence (Bolton, 1972). Logically, the puzzle is perhaps resolved if one thinks of it in terms of the difference between the capabilities to perform a task *well*, and those to perform it both well and *differently*. Originality is not valued positively within the current performance paradigms of artificial intelligence research. On the contrary, we have been concerned with systems that mimic human expertise, and our evaluations are higher the closer the performance is to established human norms. A highly creative person would not come out well within such an evaluation framework.

However, much of what has been done within the knowledge acquisition research paradigm is potentially valuable to creativity research and to the support of human creativity. The knowledge acquisition community has developed many tools designed to elicit expertise (Boose & Gaines, 1988, 1990). These are apparently within the performance framework noted above—to 'snapshot' existing expertise. However, it has been realized that, as Clancey (1989, 1990) has stressed, while the expertise may exist, the overt knowledge that we presuppose to underlie it may not pre-exist. Skilled performance in the neural network that is the brain is not necessarily based on overt knowledge structures (Gaines, 1991b). Indeed, there is no basis other than explicit presupposition to suppose that there *are* 'knowledge structures' underlying expertise.

Given this shift in perspectives, knowledge acquisition tools may be regarded, not as transferring knowledge, but rather as supporting the expert in building a model of his or her skill sufficient to emulate it. One need not presuppose that such a model pre-exists in a form to be 'transferred.' Since the roots of most knowledge acquisition tools is in other disciplines, such as psychology, anthropology and management (Shaw & Woodward, 1990), and the original purpose was often to aid people in understanding their own psychological processes, it is plausible that the computational tools developed to support conceptual model building may serve other purposes. In particular, they may be relevant to the support of the human creative process.

SUPPORTING CREATIVITY

Barron, who has authored a number of major encyclopedia articles on creativity, recently advanced what he sees as the current consensus of the nature of creativity:

"1. Creativity is an ability to respond *adaptively* to the needs for new approaches and new products. It is essentially the ability to bring something new into existence purposefully, though the process may have unconscious, or subliminally conscious, as well as fully conscious components. Novel adaptation is seen to be in the service of increased flexibility and increased power to grow and/or to survive.

2. The "something new" is usually a *product* resulting from a *process* initiated by a *person.* There are three modes in which creativity may easily be studied: as product, as process, as person.

3. The defining properties of these new products, processes, and persons, are their originality, their aptness, their validity, their adequacy in meeting a need, and a rather subtle additional property that may be called, simply, *fitness*—esthetic fitness, ecological fitness, optimum form, being "right" as well as original at the moment. The emphasis is on whatever is fresh, novel, unusual, ingenious, clever, and apt.

4. Such creative products are quite various, of course: a novel solution to a problem in mathematics; an invention; the discovery of a new chemical process; the composition of a piece of music, or a poem, or a painting; the forming of a new philosophical or religious system; an innovation in law; a fresh way of thinking about social problems; a breakthrough in ways of treating or preventing a disease; the devizing of new ways of controlling the minds of others; the invention of mighty new armaments, both of offense and defense; new ways of taxing the citizens of a country by its government; or simply a change in manners for multitudes of people in a specific period of time." (Barron, 1988)

There is *prima facie* validity of knowledge modeling tools as support for the creative process if one examines the literature on scientific development, change and revolution. For example, Torretti, in analyzing the development of modern physics, considers the dynamics of conceptual schema to be the basic framework within which all other activities can be located. In particular, he analyzes in detail the essential limits to conceptual innovation that provide continuity in scientific development, noting that:

"C1. Some concepts are immune to change, and they provide a stable reference to decisive facts.

C2. The new concepts are arrived at through internal criticism of the old, by virtue of which the facts purportedly referred to by the earlier mode of thought are effectively dissolved.

C3. Reference to facts does not depend on the concepts by which they are grasped." (Torretti, 1990)

Tools that provide means to make these relations between concepts more overt and operational should have a role in the creative process of conceptual transformation.

Barron, Torretti, and many others, describe the phenomena of creativity, but what are its underlying psychological processes? One of the first psychologists to consider the relationship between the convergent and divergent knowledge processes of creativity within a cognitive framework was George Kelly (1955) whose *personal construct theory* presents a person as exploring his or her environment, collecting data about the world and constructing a personal reality from which to view it. The repertory grid methodology used in many knowledge acquisition tools (Shaw & Gaines, 1983, 1987; Boose, 1984; Boose & Bradshaw, 1987; Diederich, Ruhmann & May, 1987) stems from this perspective, and supports the elicitation and analysis of the construct system, or conceptual framework, with which an individual operates in his or her own domain.

Kelly analyzes the dynamics of change in a person's construct system in terms of a *creativity cycle* in which convergent and divergent thinking alternate. A system of personal constructs is not fixed, but may vary from one occasion to another:

> There are typical shifts in the sequence of construction which people employ in order to meet everyday situations....The Creativity Cycle has to do with the way in which a person develops new ideas. (Kelly, 1955, pp.514-515).

Kelly sees convergence/divergence as a sequence of loosening and tightening of constructions. A person who uses only tight construing may be very productive but cannot be original. On the other hand, loose construing is characterized by a person's "preposterous thinking" (p. 529) which is unformulated and often preverbal. The structured alternation of these processes is what we perceive as the creativity underlying intelligent behavior. Thus the paradoxical relation expressed at the start of this section is explained in terms of sequentiality—the full scope of intelligence involves not only a high level of performance but the adaptive renewal of the underlying basis of that performance through a process of creative reconstruction.

Figure 1 illustrates the nature of the creativity cycle and the types of process involved in it, together with the roles that we have designed the associated computer-based tools to play in supporting these processes. Note that we see that phase of the cycle that is divergent and deconstructive as equal in importance to that which is convergent and constructive. It is as important to be able to deconstruct a well-established base of received knowledge through critical processes that can start a new creativity cycle, as it is to be able to clarify and formalize ideas into theories. This is particularly so for new graduate students where the weight of received wisdom is often the most significant burden impeding their establishing their own research directions and identities. Creativity is truly a cyclic process—one of going contrary to one's current preconceptions—but each phase of the cycle is different in nature and requires its own support. At one time one may be clarifying and substantiating one's preconceptions, at another attempting to identify and demolish them.

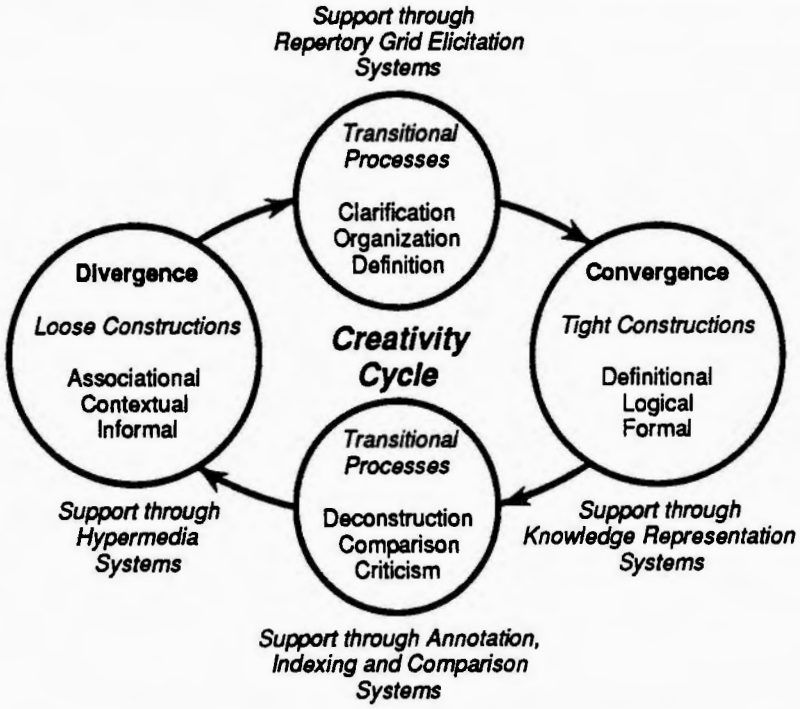


Figure 1 Creativity cycle and its support

We are studying the application of knowledge acquisition tools as systems to support the creative component of scientific knowledge processes. This paper presents Kelly's personal construct theory and, in particular, the repertory grid, as a method of eliciting relevant but abstract constructs about the topic, together with semantic nets which are used to produce a concept map

of the cognitive structures involved. Hypermedia is used for annotation and explanation, to add notes, diagrams and links to other material during the creativity process. An empirical study has been undertaken of the system in use by ten graduate students in a Cognitive Science course in a university department of Computer Science to explore and elaborate aspects of their research topics. Some initial results are presented.

KNOWLEDGE ACQUISITION TOOLS

There are well over one hundred knowledge acquisition tools named and described in the current literature (Boose, 1989). However, a recent classification of tool architectures has characterized them in terms of four major types (Gaines, 1990):

- Semi-formal elicitation and structuring through hypertext and hypermedia—used to capture raw knowledge at the pre-computational stage—the tool typically being based on a hypermedia system such as Apple's HyperCard.
- Direct editing of knowledge in a semantic network, frame, rule, representation—used with overt, computational knowledge structures directly available from an expert or derived through other tools—the tool typically being some form of graphic knowledge editor for semantic nets.
- Indirect elicitation through critical cases described in relevant attributes—used when the expert cannot provide overt knowledge structures but can provide high quality stereotypical case data—the tool typically being graphic interaction through a repertory grid methodology.
- Inductive derivation of knowledge from data sets of varying quality—used when the expert cannot provide high quality case data but can give access to substantial databases of lower quality data—the tool typically being a derivative of ID3 or AQ11.

All of these types of tools seemed to us to have some *prima facie* validity as potential support for the creative process. Our logic was that tools designed to help an expert make his or her knowledge structures overt could also support students in developing models of their own intuitions. Such tools might counterbalance the weight of existing received knowledge, particularly used by first year postgraduate students attempting to establish their own research directions and personal research identities. The significance of the knowledge acquisition tools and underlying methodologies is threefold: they are *reflective*, reflecting back to the student his or her own vocabulary, nascent knowledge structures, and their implications; they are *adaptive*, capable of being continually edited and refined with little ergonomic cost; they are *operational*, generating knowledge structures capable of being used in the computer for reasoning, simulation and problem-solving.

The support of students' learning process, in particular their personal creativity, has been a major application envisioned for hypermedia systems since their inception (Landow, 1989). The capability to access, develop and organize multiple sources and forms of knowledge is at the heart of academic scholarship. The capability to create, explore and evaluate patterns of knowledge through linking diverse materials supports the formation of the *insights* that Lonergan (1958) argued were the essence of the scientific process (Compton & Jansen, 1990). However, while the representational and linkage structures of hypermedia can support many of the argument forms significant to human discourse, they stop short of supporting the operationalization of knowledge in a logical, computable form. That, so far, has been the domain of artificial intelligence where formal knowledge representation schema support inferential reasoning processes within the computer.

However, as noted by Travers (1989), there are close relations between knowledge structures represented through hypertext webs and through semantic networks. The informal, creative richness of hypermedia and the formal, scientific precision of logic programs are not poles apart. Perhaps the human creative process of bridging between open textured ideas and closely defined theories can be supported by a technology bridging between hypermedia and semantic networks. That is the motivation underlying the system development and experiments reported in this paper.

In the early 1980's we reported the application of computer-based repertory grid tools as computer-assisted learning (CAL) tools to track the creativity cycle in a student's learning progression (Shaw & Gaines, 1982). We had already suggested that such tools could be used to derive the conceptual models underlying expertise in developing expert systems (Gaines & Shaw, 1980), and demonstrated their role in eliciting accounting expertise at an early expert systems conference (Shaw & Gaines, 1983). Boose (1984) presented parallel research in an industrial context at AAAI'84, and repertory grid methodologies and tools have since become routine items in the knowledge engineer's toolkit. In particular they have been greatly extended in both the richness of knowledge representation and the quality of interactive computer support and online computer analysis and its graphic presentation.

It has seemed to us as we enter the 1990's that the wheel has come a full circle and that it is time to test the tools developed for specialist expert system development as general *knowledge support systems* with applications to creative thinking at all stages of the educational, scientific and artistic process. We have at our disposal the original repertory grid tools, now greatly enhanced and integrated with hypermedia systems and a range of expert system shells (Gaines, Rappaport & Shaw, 1989; Gaines & Linster, 1990). We also have a graphic knowledge editor designed on the one hand for ease of use through simple MacDraw-like editing features, but on the other hand implementing a well-defined visual conceptual language supporting both Kelly's construct hierarchies and Brachman's KL-ONE conceptual structures (Gaines & Shaw, 1990). In the context of this paper, it is particularly interesting to note the way in which the graphic knowledge editor is able to support the concept maps already used within the educational system to help students elucidate and communicate their ideas (Novak & Gowin 1984; Lambiotte, Dansereau, Cross & Reynolds, 1989).

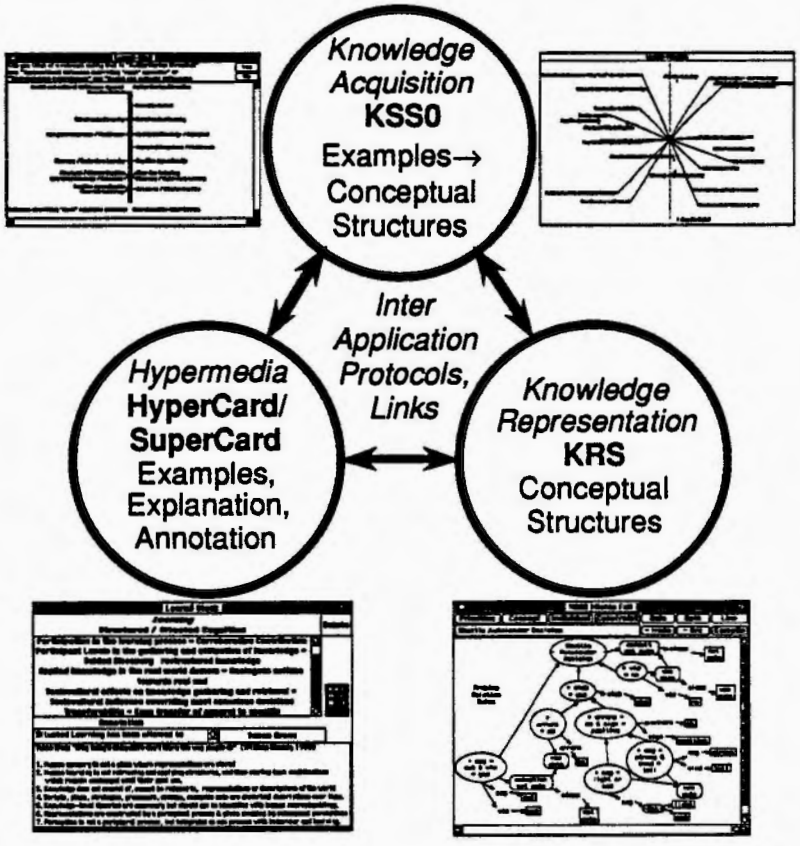


Figure 2 Integrated hypermedia, knowledge acquisition and knowledge representation

Figure 2 shows the way in which the hypermedia, knowledge acquisition and knowledge representation tools are integrated to provide an environment supporting students' creativity. The three tools communicate through inter-application protocols enabling one program to respond to the selection of actions in another. For example, examples entered in KSS0 have associated popup menus, one option of which is to transfer to an annotation card automatically created in HyperCard or SuperCard. Conversely, the grid associated with a HyperCard stack or SuperCard project can be loaded into KSS0 and clustered using commands entered through buttons in the stack or project. Similar links integrate the other applications.

The remainder of this paper gives an overview of each of the tools, illustrates their application in the hands of a graduate student, and summarizes our preliminary experience with ten students experimenting in their use.

THE HYPERTEXT TOOL

The hypermedia component of the system is the most open-architecture since both HyperCard and SuperCard provide complete programming languages and the capability to identify new user interfaces very simply and naturally. Students were encouraged to use this openness to develop interactive note books for their projects, and did so extensively, typically developing a framework for their studies and keeping logs of ideas and graphical outputs resulting from use of the other tools. One student submitted his final project as a stack with links to figures, diagrams and results as illustrated in Figure 3.

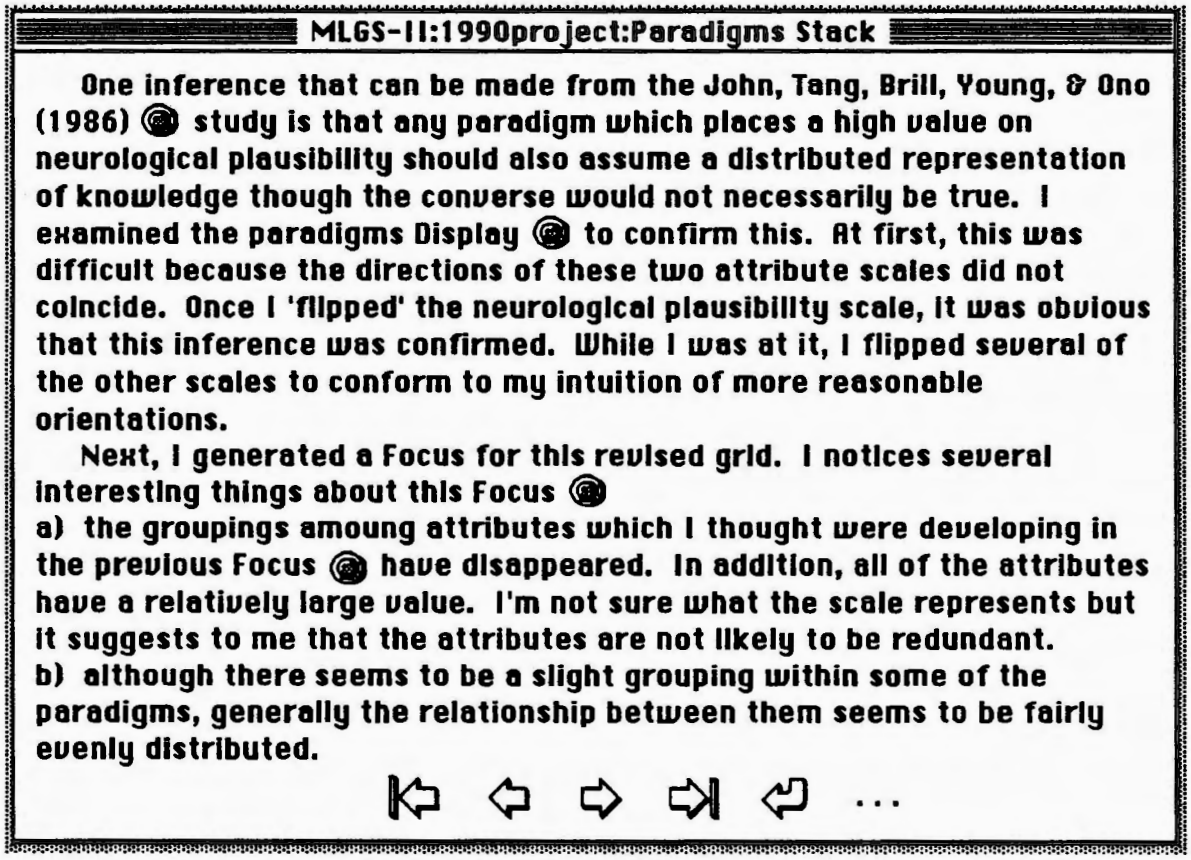


Figure 3 Card from stack submitted as final "essay" assignment

However, a completely empty stack with no initial structure is daunting, and the KSS0 tool in particular is designed to help people elicit ideas initially. Hence, students were provided with a skeleton HyperCard stack and SuperCard project that supported inter-application communication with KSS0 and KRS. When this is running at the same time as KSS0 a popup menu cursor appears as one mouses over element names and construct pole names in graphic screens such as those of Figures 5 through 9. Mousing down when this cursor appears brings up a popup menu that offers a link to the hypertext tool. Selecting this transfers to the stack or project at the card corresponding to the element or construct, creating such a card if it does not already exist.

The upper part of such annotation cards is predefined as shown in Figure 4 to show the grid name, element or construct names, and possible construct values or actual element values. The lower part contains a scrolling text area to the left for annotation and an open area to the right for buttons linking to further annotation. Facilities for linking to subject matter video disks are also provided. All the students made heavy used of the hypermedia annotation facilities to annotate their elements and constructs and to add additional material such as references, background notes and diagrams.

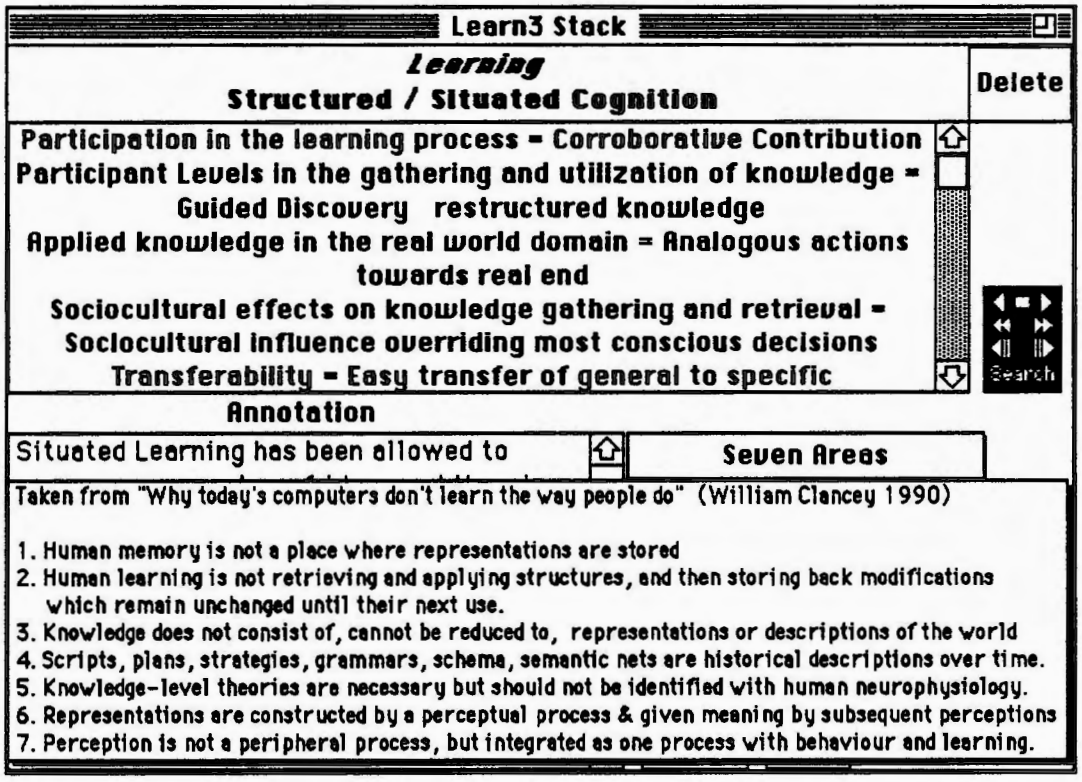


Figure 4 Element annotation card with additional annotation showing

THE KNOWLEDGE ACQUISITION TOOL

KSS0 is a knowledge support system providing an integrated set of tools for elicitation and analysis of elements and constructs in a given domain. It combines a number of different techniques, including element and construct elicitation and clustering, and is linked to an inductive rule generation program. It runs on the Apple Macintosh family of computers to provide a highly interactive and graphic knowledge acquisition environment. At the heart of KSS0 is an object-oriented knowledge base in which knowledge is formally represented as a multiple-inheritance digraph of classes, objects and properties.

The main tools in KSS0 are:

- *Elicit* accepts specifications of elements within a domain and provides an interactive graphical elicitation environment within which a person can distinguish elements to derive his or her constructs within the domain. The resultant conceptual system is continuously analyzed to provide feedback prompting the person to enter further elements and constructs.
- *Exchange* extends this to share elements and constructs between people and allows the terms in the conceptual system derived from one person to be used by another in order to determine whether the two conceptual systems are different in any way. It can also be used by the same person looking at changes in their own conceptual structures over time, for example, after reading a specific book, or exploring a particular domain.
- *Process* gives access to clustering tools for the analysis and display of the conceptual systems elicited: FOCUS shows the system as a hierarchical structure; and PrinCom as a spatial map.
- *Socio* processes results from several people to reveal the similarities and differences in their conceptual systems, or the same person at different times, construing a domain defined through common elements or constructs. It can be used to focus discussion between people on those differences between them which require resolution, enabling them to classify them in terms of differing terminologies, levels of abstraction, disagreements, misunderstandings, and so on (Shaw & Gaines, 1989).

KSS0 exports the grid data as class definitions, properties, constraints, entailments and instances to a variety of object-oriented knowledge-based system shells.

The ten graduate students were given access to KSS0 and associated tools for some two months with the assignment of developing their research ideas for their MSc research. Figures 5 through 9 give examples of the interactive graphical elicitation of constructs using KSS0 by a graduate student organizing her framework of learning strategies as exemplified by some of the publications from the Institute for Research on Learning (IRL). Figure 5 shows a construct in process of elicitation, with the elements listed on the left and being dragged on to the dimension of the bipolar construct *Facilitates Learning and Expanded Knowledge — Knowledge and Learning Hindered.*

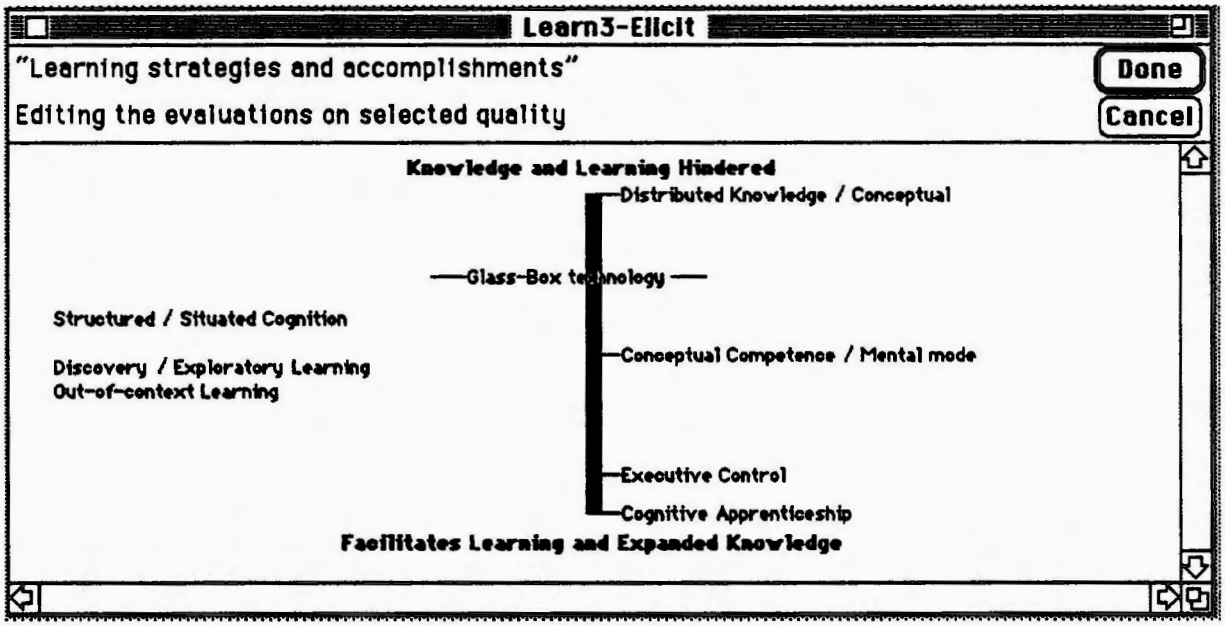


Figure 5 Construct rating during elicitation in KSS0

Figure 6 shows two constructs which are highly matched, showing the placing of the elements on each, and inviting the student to think of a new element which would reduce the match level. The popup menu is also shown, allowing the student to examine the annotation in HyperCard associated with a particular element.

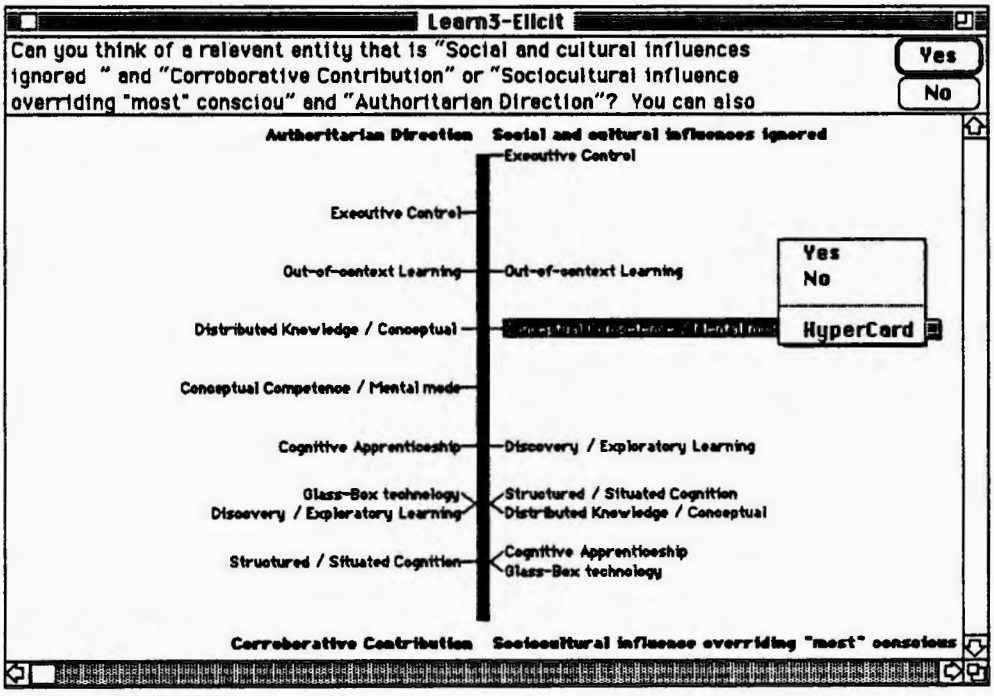


Figure 6 Construct match prompting element elicitation

Figure 7 shows a similar situation with two elements which are highly matched. The marker sticking up refers to the element *Cognitive Apprenticeship* and the marker sticking down to the other element *Structured / Situated Cognition.* Element and construct names and ratings changed at any time.

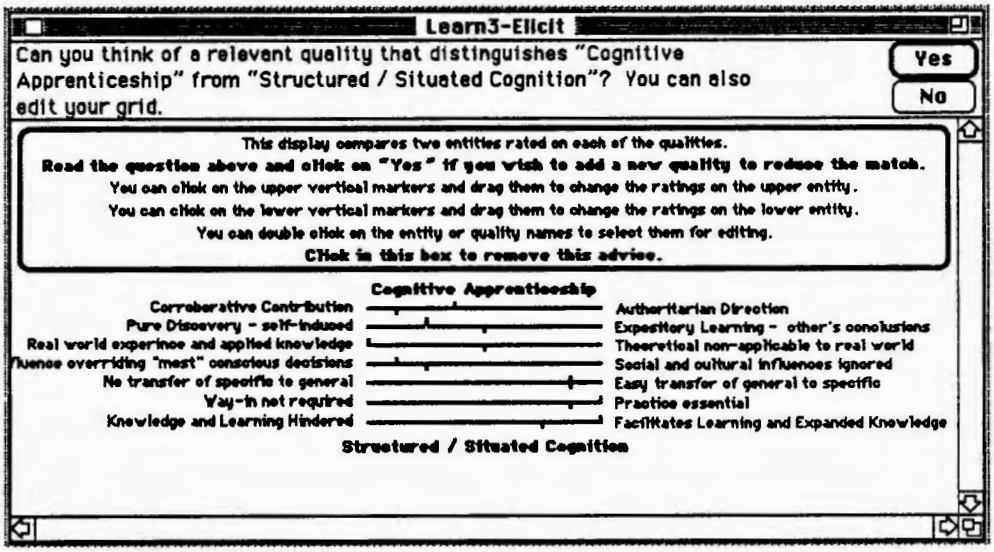


Figure 7 Element match prompting construct elicitation

KSS0 provides facilities for analyzing grids in various ways, clustering them, comparing them, deriving rules, and so on. Figure 8 shows the grid analyzed by FOCUS with the clusters on the right, and the values shaded in groups for easier interpretation, and Figure 9 shows the principal components analysis of the same data. The actual data values are also available, such as the match values and the factor loadings. The popup menu allowing access to the annotation is available for each element and construct as illustrated in Figure 9.

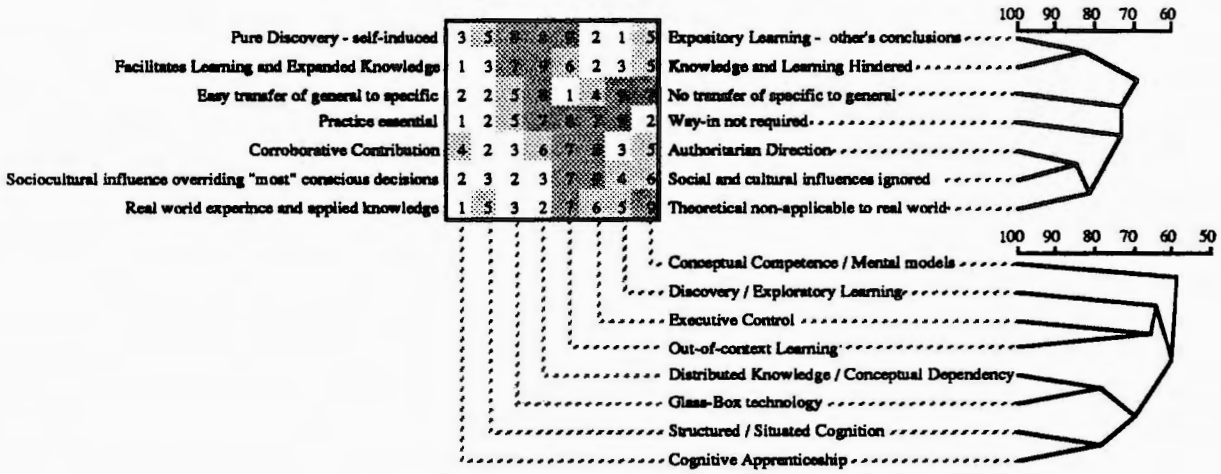


Figure 8 FOCUS hierarchical cluster analysis

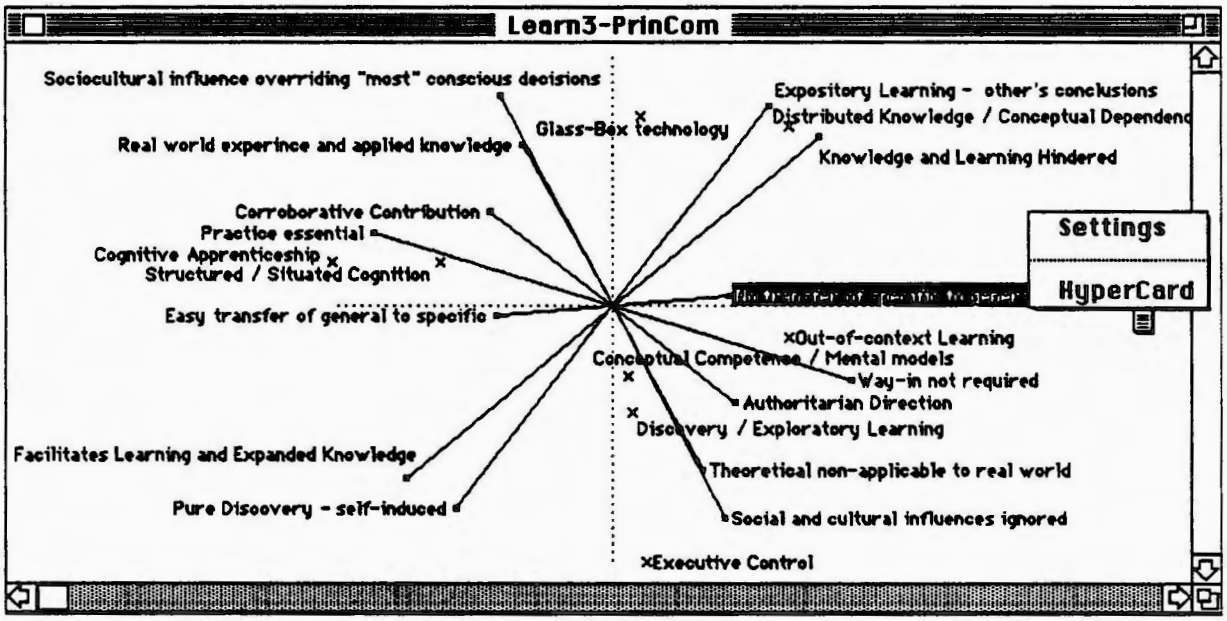


Figure 9 PrinCom spatial cluster analysis

Using the example described above the single output variable was used *Towards a Perfect System*, that is, the construct *Facilitates Learning and Expanded Knowledge — Knowledge and Learning Hindered* with the middle point described as *Limited Knowledge and Learning Gain.* Some of the rules produced by the induction system (Gaines, 1989) were:

- Knowledge is hindered when there is limited master teaching
- Knowledge is hindered when there is both cooperative participation and expository learning

- Limited knowledge gain occurs in purely theoretical environments
- Pure discovery facilitates knowledge gain
- Analogous learning supported by applied practice facilitates learning

The knowledge structures and rules developed in KSS0 may be exported as classes, objects and rules to a number of knowledge-based system shells to become overt, operational knowledge structures. This was not done in the present study, largely through lack of time and the complexity of adding yet another tool to the set that the students were involved in using. However, the creation of an operational knowledge structure may be seen as a crucial test of the convergence of ideas to a simulable form where the consequences of assertions may be inferred and tested, and the would be a useful component of future studies.

KSS0 allows grids and annotation developed by multiple users to be compared (Shaw & Gaines, 1989). One of the students asked his supervisor to create a grid using the elements and constructs from his own grid on instructible systems. Figure 10 shows a comparison of these two grid. The elements and constructs on which they agree and differ are immediately apparent and can form the focus of productive discussion; the darker shaded areas toward the bottom left showing the most conflict. Students may compare their views among themselves and with faculty. The basis of differing opinions is clearly apparent and open to discussion.

For example, the student and supervisor were in consensus on the top six constructs but had a significant conflict in their use of the bottom three constructs. Similarly, they agreed about the four top elements but had very different perspectives on the bottom one. It is valuable to focus on the conflicts and disagreements and determine their sources since differing uses of the same terminology, or different viewpoints on the same systems, may lead to misunderstanding based on tacit presuppositions.

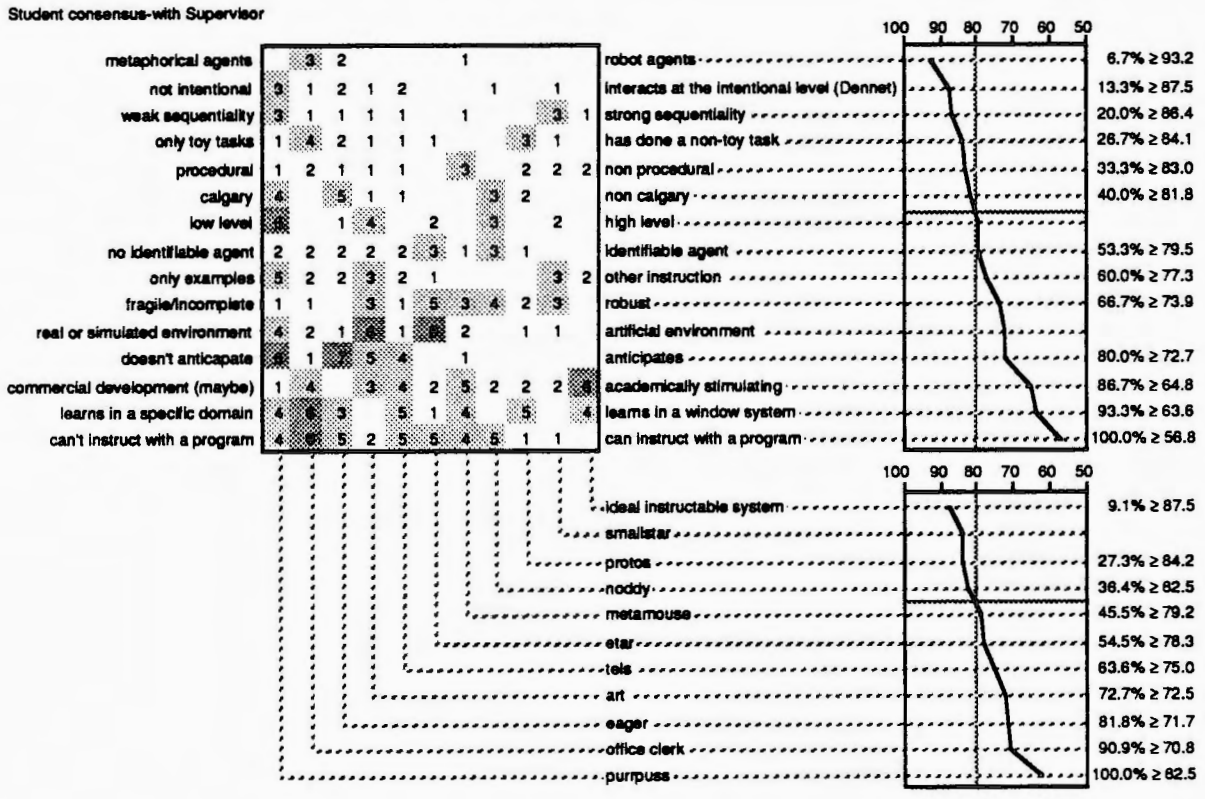


Figure 10 Comparing exchanged grids from student and supervisor

THE KNOWLEDGE EDITING TOOL

The knowledge structures developed in KSS0 can be exported to a variety of knowledge-based system shells as class, property, constraint, object, value and rule definitions. In particular they may be exported to the graphic editing tool associated with our knowledge representation server, KRS (Gaines, 1991a). Figure 11 shows the grid concepts and rules shown above exported to KRS and graphed as a semantic net.

Term subsumption, or KL-ONE-style, knowledge structures are notoriously difficult to edit because they are require the 'top-down' definition of general concepts and their stepwise refinement to more specific concepts. The grid methodology is a way of developing them 'bottom up' from cases. The KRS graphic knowledge editing tool further supports this approach (Gaines, 1991c). Knowledge structures can be developed and edited using drawing tools providing a natural visual language. The structures can extend across multiple screens supporting modularity and libraries. Structures can be freely duplicated and definitions may be fragmented across screens. The consistency of the definitions is determined only when they are compiled into a formal knowledge structure and inconsistencies are graphically highlighted.

The grapher is highly interactive. Objects, and groups of objects, may be dragged to new locations and the lines remain connected. Objects may be double-clicked to select them for editing—both text and object type may be edited. Popup menus associated with each object enable the connecting lines and arrows to be entered simply.

The visual language used in Figure 11 is precisely defined. Concepts are ovals, primitive concepts are ovals with small horizontal lines inside each side, individuals rectangles, roles (and annotation) unboxed text, rules rounded-corner boxes, constraint expressions rounded-corner boxes with small horizontal lines. The interpretation of the arrows is overloaded but well-defined by the types of the objects at their head and tail. Lines without arrows connecting primitive concepts denote that the concepts are disjoint, and connecting roles that they are inverse. The knowledge compiler traces concepts and individuals through their outgoing arrows, builds a dependency table for concepts, highlighting circular links that it will not use, and generates concept definitions and individual assertions which are passed to the server.

All of the students exported their grid data to KRS and graphed it from various perspectives. One student, who was very much taken by the graphic editing tool in its own right, used it directly to develop a number of different classifications of material in his research area of instructible systems in artificial intelligence. Figure 12 shows some concept maps produced by this student. It is noteworthy that he used KRS more as a concept mapping tool rather than as a formal visual language for term subsumption knowledge structures expressed in a visual language. He remarked that he preferred the visual language of the graphic knowledge editor to the textual approach of an outliner.

Ausubel (1969) proposed concept maps as a tool in education for both developing students' creativity and assessing their changing conceptual structures during the learning process. Novak & Gowin (1984) and Lambiotte, Dansereau, Cross & Reynolds (1989) have reported extensive use of concept maps in the educational system. Hunter, Stahl & Novak (1990) have developed CMap, a computer-based conceptual map tool. The graphic editor in KRS may be seen as a rather more elaborate and formally based version of such a tool, and it would be interesting to see its impact in use by those in the educational system already using concept maps.

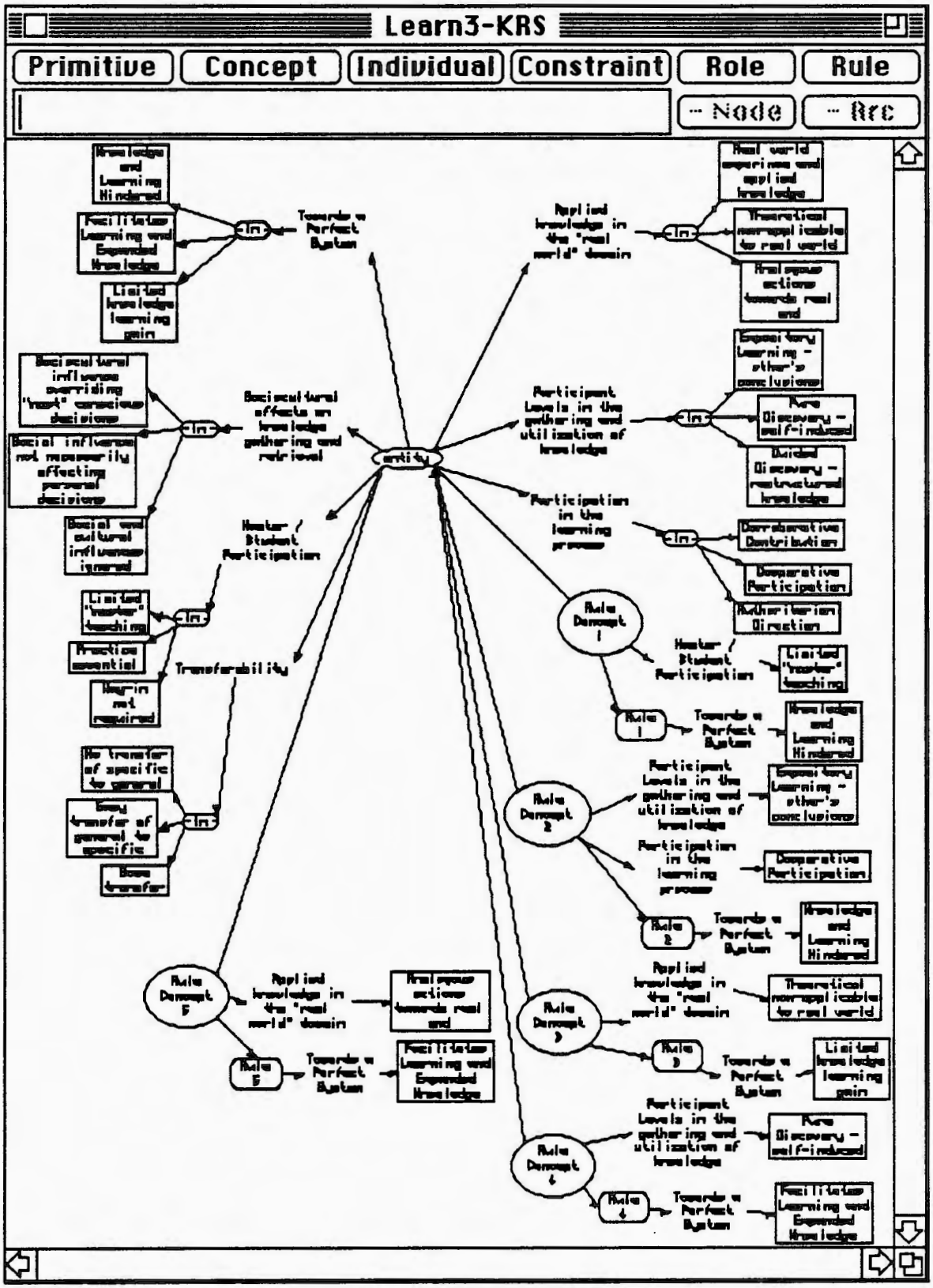


Figure 11 Grid and induced rules exported to the KRS graphic knowledge editor

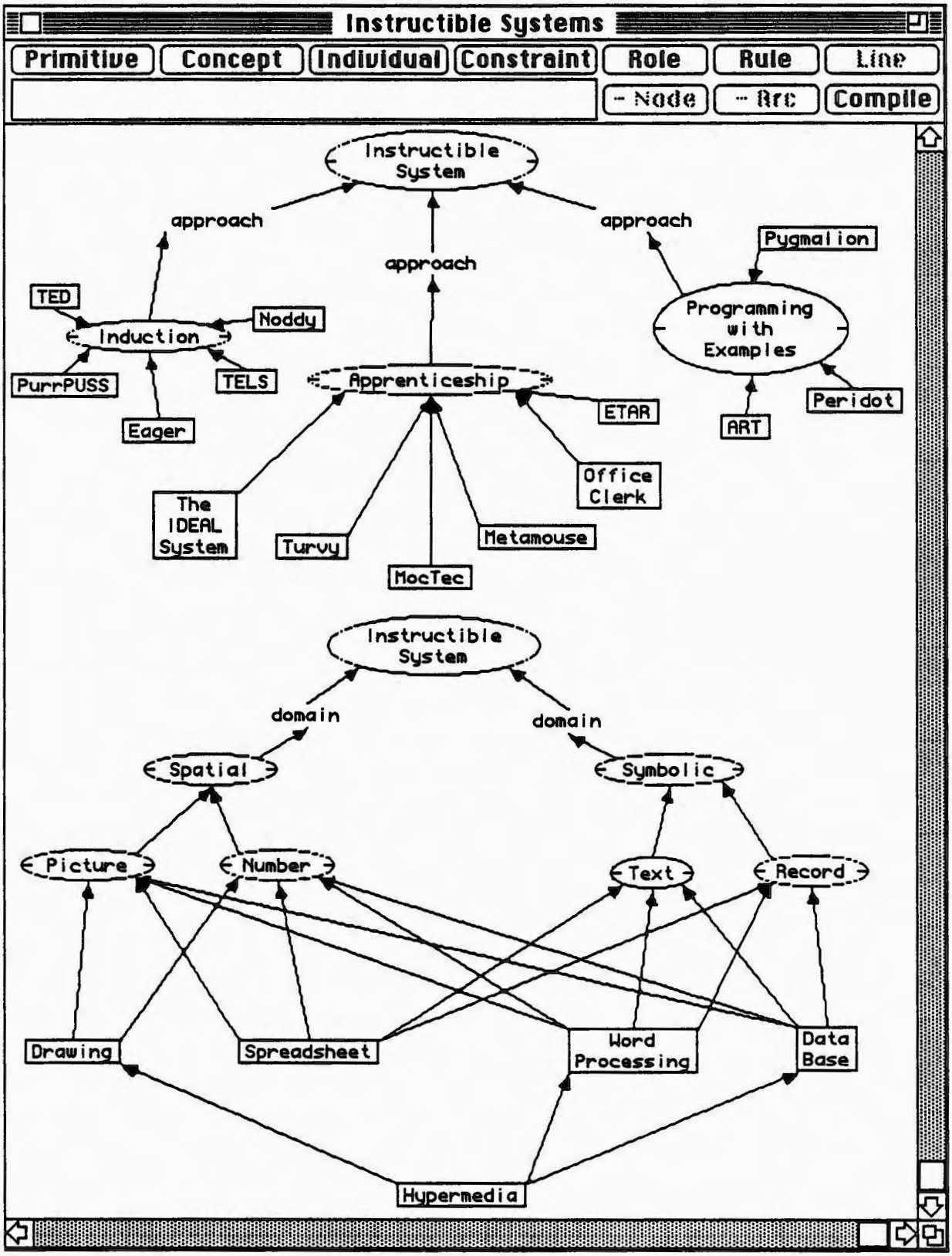


Figure 12 Concept maps produced by student researching instructible systems

SOME RESULTS

The students in the Cognitive Science course were encouraged to choose a topic which was closely related to their area of interest and their research project. We have already seen one example in the general area of situated cognition, and another in instructible systems; others included theories of mind, psychological paradigms and induction systems. Figure 13 summarizes the research topics, domain chosen for elicitation, outcome in terms of material produced, and major comments of the students on the course.

	Research Topic	Chosen Domain	Outcome	Student Comments
1	open	learning strategies	grid, knowledge structure, stack	
2	open	situated cognition	grid, knowledge structure, stack	
3	instructible systems	instructible systems	2 grids, exchange with Socio analysis, 7 knowledge structures, stacks	prefer KRS to an outliner for developing ideas
4	expository writing support tools	psychological paradigms	2 grids, knowledge structures, stack	presented results as a HyperCard stack
5	cognitive science	theories of mind	grid, knowledge structure, stack	would like to use this for my thesis
6	intelligent tutoring	Pascal statements	grid, stack	
7	open	computational learning	grid, knowledge structure, stack	
8	genetic algorithms	computer games	grid, knowledge structure, stack	
9	machine learning	induction systems	grid, knowledge structure, stack	learned a lot more about induction systems doing this sort of structuring
10	complexity of computation	fast food	grid, knowledge structure, stack	concepts in research field are already defined and not suitable

Figure 13 Summary of student experience

It can be seen that not all the students chose topics associated with their research, and it is suspected from more detailed discussion that this was a form of cognitive defense in the same way that avoidance of a primary topic is in a clinical context. There was no pressure to conform since, from an educational point of view, we were primarily concerned to introduce the students to tools within a cognitive science framework that modeled conceptual structures. The fact that they could be used for a variety of practical purposes within the lifeworlds of the students was incidental. The tools continue to be available in the Department and it will be interesting to see whether they are further used by the students concerned in the future development of their research topics.

CONCLUSIONS

What has been reported here is very much a preliminary, informal investigation of the application of tools developed within the knowledge acquisition community to support the creative process in education. Although the underlying methodologies go back many years, it is only in very recently that these tools have reached a level of usability, and deliverability on personal computers, that makes it feasible to introduce them into the educational system. There is still much to be done in the total 'packaging' of such tools for the non-AI community.

It would be very interesting as knowledge acquisition tools mature to attempt to make them available as 'conceptualization' support to the scientific community. Even though from the knowledge acquisition community perspective we still have far to go in developing theories,

methodologies and tools, and anything we have today has a poor resemblance to our aspirations, it may not be too early to investigate potential applications in the sciences. One sees many examples of scientific thinking where existing tools would help clarify the arguments.

For example, the economist Weintraub, has recently published a methodological analysis of the way in which economic knowledge is constructed and used this to study the historical development of general equilibrium theory. One of his major conclusions is:

> "The main argument that develops from this set of observations is that the literature on dynamic stability, as it developed from the 1930s to the 1950s, was formed and guided by a set of 'oppositions,' and those oppositions themselves created the language and conceptual framework that structure how modern theorists think about the economy and the way we model-construct the economy." (Weintraub, 1991)

He develops these oppositions in detail, and exhibits them and their interplay in the course of economic history. It seems likely that this research, and its presentation, would benefit immediately from the use of a repertory grid tool to analyze economic phenomena and research in terms of these oppositions.

Such types of analysis are found in many disciplines. In musicology, for example, Cogan (1984) develops a detailed analysis of the tone qualities of musical sound in terms of a set of some thirteen binary oppositions. He characterizes the changing tone structure of a number of pieces of music as they develop chronologically, and uses this to compare pieces. However, the analysis is descriptive and makes no further use of the material available which again fits neatly into a repertory grid framework.

Cruse's (1986) book on linguistic semantics provides many examples where a knowledge modeling tool could be effective in exemplifying and strengthening the arguments. For example, he argues that *father* and *mother* are "weaker opposites" than *man* and *woman* which, in turn, "are weaker than" *male* and *female*. However, the argument is highly discursive and nether logical nor quantitative. If we graph the knowledge structures involved as shown in Figure 14, it can be seen that, for example, *father* and *mother* inherit the disjoint relation between *male* and *female*, but also inherit in common the attributes of *person* and *parent*. Thus, Cruse's analysis of the relative degree of opposition in terms of commonality is apparent in this representation.

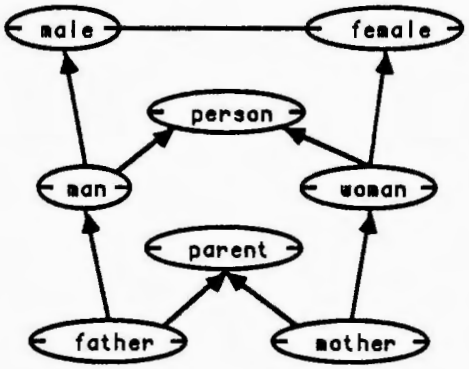


Figure 14 The relative strength of opposition apparent in a knowledge structure

Feldman, Compton and Smythe (1989) gave an example of the application of a knowledge-based tool to hypothesis testing in supporting clinical scientists developing models of a biochemical system. The tool JUSTIN was able to analyze the support or counter-indications for structural or causal hypotheses. It would be very interesting to see such tools developed more generally and made widely available in the sciences.

Thadgard has published a range of papers addressing the modeling of the phenomena of conceptual change in the sciences outlined at the beginning of this paper. For example, Thadgard and Nowak (1990) develops a very detailed analysis of the intellectual history of plate tectonics, and attempts to account for the dynamics of the early rejection and later acceptance of the modern theories. Thadgard uses a neural net model to show coherence of theories through spreading activation. This seems to give meaningful global assessments of the relative attractiveness of competing hypotheses, but is not perspicuous in showing how change occurs at the knowledge level. It would be interesting to reconstruct his analyses using knowledge modeling tools.

In conclusion, we see the support of the human creative process and the support of experts in forming overt models of a basis for their skilled performance as closely related topics. Kelly's model of creativity puts it strongly in cognitive terms that gives formal foundations to this view. Even if this is only part of the story and there are dimensions to creativity well outside the cognitive paradigm, there seems to be enough within it to make the enterprise worthwhile.

ACKNOWLEDGEMENTS

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada. We wish to thank the 1990 class of CPSC 679, *Cognitive Processes in Artificial Intelligence.*

REFERENCES

Ausubel, D.P. & Robinson, F.G. (1969). **School Learning.** New York: Holt, Rinehart and Winston.

Barron, F. (1988). Putting creativity to work. Sternberg, R.J., Ed. **The Nature of Creativity.** pp. 76-98. Cambridge: University Press.

Bolton, N. (1972). **The Psychology of Thinking.** London: Methuen.

Boose, J.H. (1984). Personal construct theory and the transfer of human expertise. **Proceedings AAAI-84,** 27-33. California: American Association for Artificial Intelligence.

Boose, J H. (1989). A survey of knowledge acquisition techniques and tools. **Knowledge Acquisition 1** (1), 39-58 (March).

Boose, J.H. & Bradshaw, J.M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. **International Journal of Man-Machine Studies, 26,** 3-28.

Boose, J.H. & Gaines, B.R., Eds. (1988). **Knowledge Acquisition Tools for Expert Systems.** London, Academic Press.

Boose, J.H. & Gaines, B.R., Eds. (1990). **The Foundations of Knowledge Acquisition.** London, Academic Press.

Clancey, W. (1989). Viewing knowledge bases as qualitative models. **IEEE Expert,** 4(2), 9-23.

Clancey, W.J. (1990). The frame of reference problem in the design of intelligent machines. In K. van Lehn & A. Newell, **Architectures for Intelligence: The Twenty-Second Carnegie Symposium on Cognition.** Hillsdale: LEA.

Cogan, R. (1984). **New Images of Musical Sound.** Cambridge, Massachusetts: Harvard University Press.

Compton, P. & Jansen, R. (1990) A philosophical basis for knowledge acquisition. **Knowledge Acquisition** 2(3).241-258.

Cruse, D.A. (1986). **Lexical Semantics.** Cambridge: University Press.

Diederich, J., Ruhmann, I. & May, M. (1987). KRITON: A knowledge acquisition tool for expert systems. **International Journal of Man-Machine Studies, 26**(1), 29-40.

Feldman, B.Z., Compton, P.J. & Smythe, G.A. (1989). Hypothesis testing: an appropriate task for knowledge-based systems. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.10-1-10-20. Banff (October).

Gaines, B.R. (1989). An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. **Proceedings of 6th International Workshop on Machine Learning,** pp.156-159. San Mateo, California: Morgan Kaufmann (June).

Gaines, B.R. (1990). An architecture for integrated knowledge acquisition systems. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.8-1-8-22. Banff (November).

Gaines, B.R. (1991a). Empirical investigation of knowledge representation servers: design issues and applications experience with KRS. **AAAI Spring Symposium: Implemented Knowledge Representation and Reasoning Systems.** pp. 87-101. Stanford (March).

Gaines, B.R. (1991b) Between neuron, culture and logic: explicating the cognitive nexus. **Proceedings of ICO'91: Cognitive Science: Tools for the Development of Organizations.** pp. 222-32. Montreal: GIRICO.

Gaines, B.R. (1991c). An interactive visual language for term subsumption visual languages. **IJCAI'91: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence.** San Mateo, California: Morgan Kaufmann.

Gaines, B.R. & Linster, M. (1990). Development of second generation knowledge acquisition systems. Wielinga, B., Boose, J.H. & Gaines, B.R., Schreiber, G., van Someren, M., Eds. **Current Trends in Knowledge Acquisition.** pp. 143-160. Amsterdam, IOS.

Gaines, B.R., Rappaport, A. & Shaw, M.L.G. (1989). A heterogeneous knowledge support system. Boose, J.H. & Gaines, B.R., Eds. **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.13-1-13-20. Banff (October).

Gaines, B.R. & Shaw, M.L.G. (1980). New directions in the analysis and interactive elicitation of personal construct systems. **International Journal of Man-Machine Studies, 13,** 81-116.

Gaines, B.R. & Shaw, M.L.G. (1990). Cognitive and logical foundations of knowledge acquisition. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff (November).

Hunter, S., Stahl, H.A. & Novak, J.D. (1990). CMap: a concept mapping elicitation tool. **Technical Report #90-006,** Department of Education, Cornell University.

Kelly, G.A. (1955). **The Psychology of Personal Constructs.** New York: Norton.

Lambiotte, J.G., Dansereau, D.F., Cross, D.R. & Reynolds, S.B. (1989). Multirelational semantic maps. **Educational Psychology Review, 1**(4), 331-367.

Landow, G.P. (1989). Hypertext in literary education, criticism and scholarship. **Computers and the Humanities, 23,** 173-198.

Lonergan, B.J.F. (1958). **Insight: A Study of Human Understanding.** New York: Harper & Row.

Novak J.D. & Gowin D.B. (1984). **Learning How To Learn.** New York: Cambridge University Press.

Shaw, M.L.G. & Gaines, B.R. (1982). Tracking the Creativity Cycle with a Microcomputer. **International Journal of Man-Machine Studies, 17**(1), 75-85 (July).

Shaw, M.L.G. & Gaines, B.R. (1983). A computer aid to knowledge engineering. **Proceedings of British Computer Society Conference on Expert Systems,** 263-271 (December). Cambridge.

Shaw, M.L.G. & Gaines, B.R. (1987). KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. **International Journal of Man-Machine Studies, 27,** 251-280.

Shaw, M.L.G. & Gaines, B.R. (1989). A methodology for recognizing conflict, correspondence, consensus and contrast in a knowledge acquisition system. **Knowledge Acquisition 1**(4), 341-363 (December).

Shaw, M.L.G. & Woodward, J.B. (1990). Mental models in the knowledge acquisition process. **Knowledge Acquisition: An International Journal,** 179-206.

Thadgard, P. & Nowak, G. (1990). The conceptual structure of the geological revolution. Shrager, J. & Langley, P., Eds. **Computational Models of Scientific Discovery and Theory Formation.** pp. 27-65. san Mateo: Morgan Kaufmann.

Torretti, R. (1990). **Creative Understanding: Philosophical Reflections on Physics.** Chicago: University of Chicago Press.

Travers, M. (1989). A visual representation for knowledge structures. **Proceedings of HYPERTEXT'89** 147-158.

Weintraub, E.R. (1991). **Stabilizing Dynamics: Constructing Economic Knowledge.** Cambridge: University Press.

EXPANDING THE DESCRIPTIVE POWER OF GUIDED SELF-ELICITATION

Glenn G. Shephard, College of Business, San Jose State University, One Washington Square, San Jose, CA 95192-0069

ABSTRACT

Recent exploratory research developed and tested a guided self-elicitation (GSE) methodology. With GSE, an expert is enabled to capture his/her own performed expertise as production rule-instances. GSE is based on published cognitive research, using a production system view of conscious cognitive information processing and certain demonstrated human abilities: for identifying and categorizing perception, rehearsing and reconstituting prior thought processes and verbal reporting of concurrent cognitive information processing. Experimentally self-elicited decision analyst expertise (leading subjective probability assessment interviews) demonstrates that performed expertise can consist of complex rule-processed knowledge forms. An object model for representing complex knowledge forms is proposed and discussed.

INTRODUCTION

A knowledge engineer (KE) conventionally collects data about human domain expertise through observation and inquiry of a recognized performing expert. The KE independently analyzes, interprets and composes the data into a knowledge structure supposedly representative of the human expert's task performance expertise. This "engineered" knowledge is then used to create expert systems intended to perform on a level comparable to the recognized human expert.

Task observation can only provide externally perceivable indications of ongoing cognitive processing activity. Internal indicators are unavailable. Introspective/retrospective reporting by an expert is suspect. To the extent that performed expertise is complex and difficult to comprehend, important differences may exist between engineered task knowledge and that actually used by a performing human expert. Such differences may materially impact the quality of expert system performance.

GSE methodology approaches knowledge elicitation in a manner reversed from that conventionally employed. Rather than relying on the KE to develop an expert knowledge base from collected data about expertise, GSE seeks to enable a performing expert to self-elicit his/her own performed expertise through a process of "guided self-elicitation." The KE's role in this process is to

transfer cognitive processing and self-elicitation process knowledge to the expert for the expert's own use in reporting his/her own task directing cognitive processing.

Published research findings were used to develop a framework model of conscious cognitive information processing for GSE. The model was instrumental in designing the GSE methodology and an exploratory investigation of it. The domain selected for study was subjective probability assessment. The expert knowledge addressed in the study was the guiding expertise of a consulting decision analyst performed while leading a business person to probabilistically encode an uncertain business variable during an assessment interview. The experimentation demonstrated the viability of the GSE approach and captured the actual performed expertise of an experienced decision analyst, practiced in three assessment interviews, as production system rule-instance traces.

Self-elicited guidance expertise for probability assessment interviewing has shown itself to be complex and multifaceted. The experienced decision analyst seems to reason quantitatively with probability requirements, qualitatively to compensate for the biasing affects of cognitive heuristics and heuristically with experienced-based interview pacing; all in orchestrated combination. Initial GSE guidance protocols proved incomplete and hampered the decision analyst's description of the structures and forms of his internal knowledge. An object model has been proposed to represent arbitrarily complex forms of knowledge that may be used by experts in rule-based processing. Current research is aimed at testing the descriptive power of this GSE object model to adequately describe human decision analyst expertise.

INITIAL GSE METHODOLOGY

Verbal reporting of prior cognitive processing has been shown to be untrustworthy [Nisbett and Wilson, 1977][Ericsson and Simon, 1980,1984]. And knowledge "engineered" from such sources should be viewed with suspicion [Vandamme, 1987]. However, this problem may be obviated if prior cognitive information processing can be refreshed. GSE is based on the idea that retrospective prompting with an expert's own perceptions, recorded during task performance, can serve to rehearse (rote rehearsal per [Zechmeister and Nyberg, 1982]) and reconstitute the stream of prior cognitive information processing for authentic reporting by the expert.

The GSE approach employs modern videotape recording and editing methods to assist the expert in self-eliciting his/her own performed expertise. Videotape is used to record both the visual and aural perceptions of the expert during task performances. The recordings provide the basis for creating and

labeling "expertise review tapes" (ERTs). They are used by the expert to rehearse and cognitively relive his/her recorded performances. Using KE-prepared self-elicitation protocols, the expert performs a series of focused exercises to identify, label, interpret and associate his/her recorded perceptions into performance traces of rule-instances.

Published cognitive psychological research underlies the knowledge elicitation transfer protocols used in GSE. A simple qualitative model of conscious, rule-based cognitive information processing was developed for expert use with GSE. It was an easily understood framework, a comfortable metaphor for the experts involved in the research and was derived from the standard Information Processing System (IPS) model [Newell and Simon, 1972]. It is shown in Figure 1 as an adaptation of the Short Model (Gilmartin, Newell and Simon, 1976].

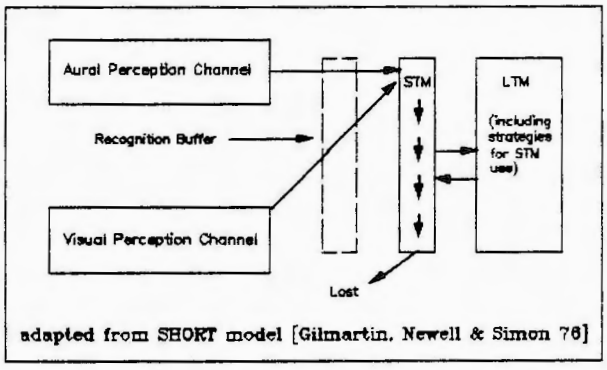


Figure 1. Conscious, Rule-Based Cognitive Information Processing

In this qualitative model, meaningful information structures are identified in the flow of visual and aural perception during task performance as categorized entities [Bruner, Goodnow and Austin, 1956]. The entities are viewed by the model as discrete information units subject to cognitive processing. GSE calls these entities knowledge elements. There can be both external, perceptual knowledge elements and internal, non-perceptual knowledge elements. Whichever their source, knowledge elements enter a finite IPS short term memory (STM) queue as symbols - which pass through and are lost [Newell and Simon, 1972, 1976]. Perceptual knowledge elements enter the IPS model as stimulus pattern signals through the visual and aural perception channels. They pass through the recognition buffer where they are assigned symbol identities which enter the STM queue. Internal knowledge elements are generated through the internal thought processes of the expert and do not involve perceptual inputs. But they do contribute knowledge elements as symbols to the STM queue.

Self-elicitation of thought processes by an expert requires reporting. But the ability to report ongoing cognitive information processing is restricted. People are able to concurrently report their actual cognitive information processing but are not able to do so retrospectively or introspectively [Nisbett and Wilson, 1977][Ericcson and Simon, 1980,1984]. This reporting asymmetry triggered the basic notion of GSE; that retroactive prompting of prior cognitive information processing could enable concurrent reporting.

GSE methodology requires the participating expert to iterate through a series of focused ERT review exercises. Each GSE exercise requires a different review focus where the expert is directed to attend specifically to one aspect/stream of his/her rehearsed thought processes - excluding all others. Such single-minded focusing of expert attention to one specific part of his/her ongoing processing enables the expert to verbally report on this aspect of reasoning [Anderson, 1990][Ericsson and Simon 1980,1984].

The final feature of the GSE-IPS model is the limited queue size of STM [Miller, 1956]. Queue capacity is limited to no more than "seven plus or minus two" compiled information "chunks" at any time. (Compiled information "chunks" may correspond closely to GSE knowledge elements.) GSE uses this limitation to provide some basis for judging knowledge element granularity of self-elicited expertise.

EXPERIMENTAL DEVELOPMENT AND TESTING

GSE was used in experimental research to record and analyze the performed natural expertise of an experienced decision analyst acting as a consultant to lead a series of subjective probability assessment interviews. During each assessment interview, the same decision analyst led a different business task expert to accurately encode one of his own relevant task uncertainties as a formal probability distribution. The analyst helped each business expert overcome the cognitive biases inherent in conventional thinking about his subjective feelings toward the uncertainty [Tversky and Kahneman, 1974]. The GSE methodology, tailored for an interviewing process, was storyboarded to assist explanation of videotape recording and the GSE exercises.

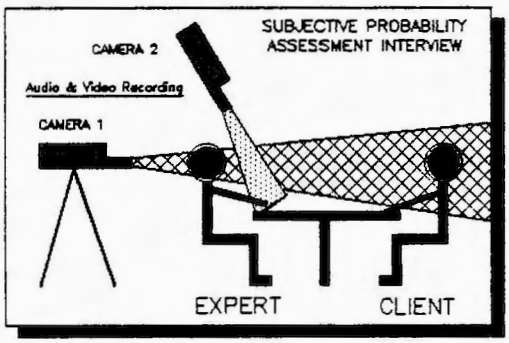
Three subjective probability assessment interviews with three different production planners who worked in the same department of a large Aerospace Company were subjected to GSE. One uncertain variable was assessed in each interview; a different, but similar variable for each. The procedure used by the decision analyst was a slightly modified version of published SRI/SDG protocols for subjective probability assessment [Merkhofer, 1987][Stael von Holstein and Matheson,

1978][Spetzler and Stael von Holstein, 1975]. The decision analyst led the assessor clients by (1) procedurally directing the encoding process and (2) assisting the clients in compensating for their displayed effects of cognitive heuristics normally employed during the encoding process.

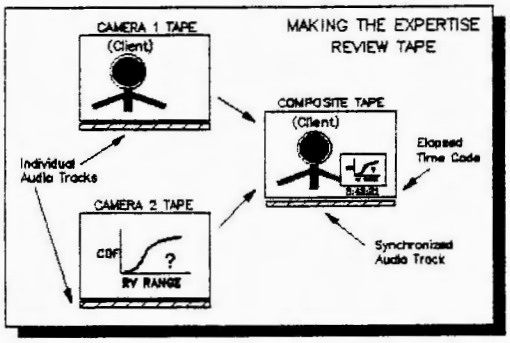
Published SRI/SDG protocols provide an agenda for decision analyst consultants to use when leading subjective probability assessment interviews [Merkhofer, 1987]. The participating decision analyst used a slightly expanded and revised version of the SRI/SDG agenda:

(1) Introduction	
(2) Motivation	- SRI/SDG Stage 1
(3) Structuring	- SRI/SDG Stage 2
(4) Conditioning	- SRI/SDG Stage 3
(4) Pre-Encoding Practice	
(6) Encoding	- SRI/SDG Stage 4
(7) Verification	- SRI/SDG Stage 5

All three interviews were set up and run as shown in Storyboard 1. Sets of videotape recordings of decision analyst visual and aural perceptions during the interviews were used to prepare ERTs for expert GSE exercises. This is shown in Storyboard 2.



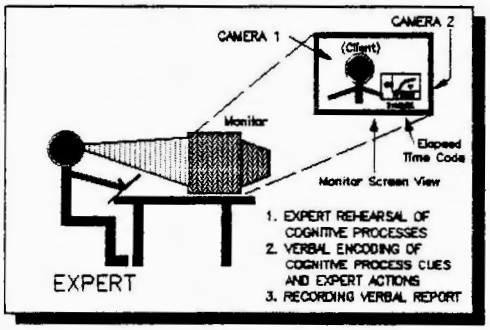
Storyboard 1



Storyboard 2

The experimental ERTs were used by the decision analyst expert to rehearse and reconstitute his prior ongoing cognitive processes actually used when guiding each assessor client. The first step in the GSE process is identification-labeling. In the identification-labeling exercise, the analyst transduced and translated his videotaped interview perception patterns (visual and aural) into videotape frame sequence blocks of labeled external cues and personal interventions (actions). The process is shown in Storyboard 3.

The decision analyst recorded his identification-labels on a prepared worksheet. An example is shown in Figure 2. It provided space for recording current operating goal, start time of next cognitive processing element and short descriptions for external perception cues and analyst interventions.



Storyboard 3

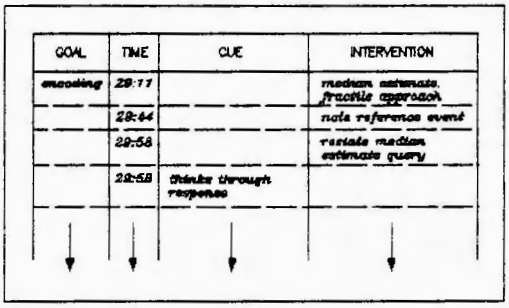
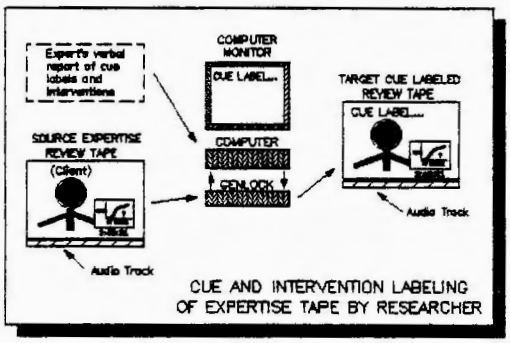


Figure 2.
Identification-Labeling
Worksheet

From dialogue transcribed from the ERTs and the analyst's worksheets, formal written protocols of the interview performances were created. The dialogue of each interview was used as a proxy for the decision analyst's full videotape recorded visual and aural perceptions. The analyst's labels were also overlaid onto the ERTs as shown in Storyboard 4. The resulting screen display is shown in Figure 3. The new recorded copies are called "labeled expertise review tapes" (LERTs).



Storyboard 4

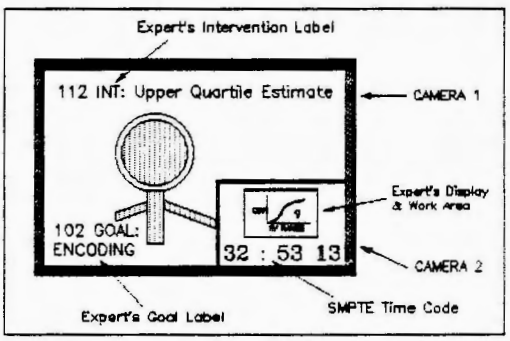


Figure 3. Labeled Expertise
Review Tape Screen

After the LERTs were created, the analyst viewed them again and recognized that another level of knowledge processing was operating during the interviews; concurrent with that already reported. The identification-labeling exercise was repeated by the decision analyst.

The second identification-labeling exercise produced additional labeled knowledge elements relating to the decision analyst's "pacing" of the assessment interviews. These were added to the existing expertise performance protocols and LERTs. Figure 4 shows the new identification-labeling worksheet developed by the analyst and Figure 5 shows an updated, "fully labeled expertise review tape" (FLERT) screen.

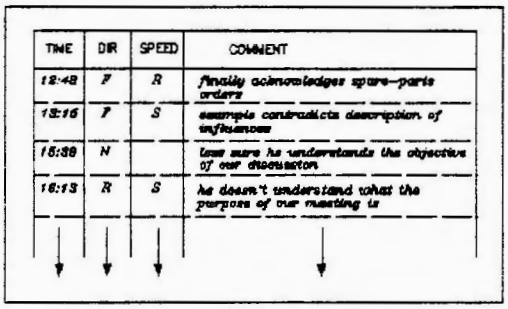


Figure 4. Pacing Identification-Labeling Worksheet

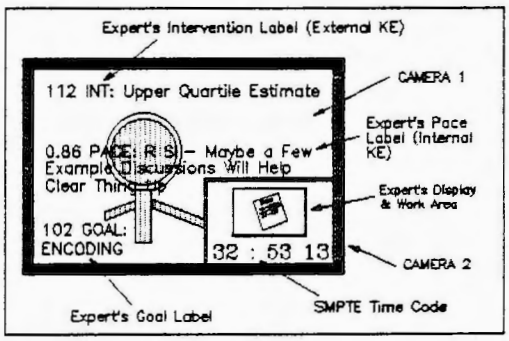
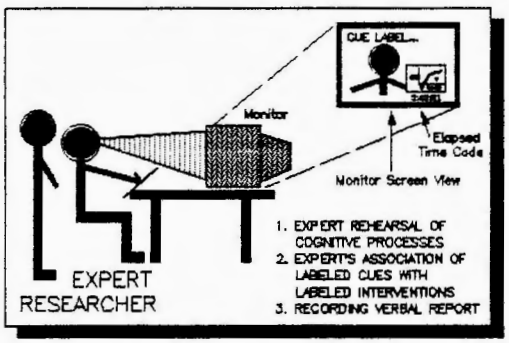


Figure 5. Fully Labeled Expertise Review Tape Screen

The second step in GSE is the knowledge element association exercise. It required the decision analyst to rehearse his FLERT and to record his associations on the fully labeled expertise performance protocols. This is illustrated in Storyboard 5.



Storyboard 5

```
IF    0.12 PACE: SHIFT TO FORWARD/SLOW
           (Expert Comment: Things are
            slowing down.)
      16   GOAL: STRUCTURING
      32   INT: CLARITY TEST FOR VARIABLE
      33   CUE: VARIABLE NOT CLEAR
      34   INT: REDEFINE VARIABLE

THEN  35   INT: CLARITY TEST
```

Figure 6. Decision Analyst's Formulated Rule-Instance Association

During the association exercise, the analyst identified groups of prior knowledge elements that were causally associated and

responsible for each rule-instance's execution and observed intervention. Figure 6 shows an example of one of the decision analyst's knowledge element associations.

DECISION ANALYST EXPERTISE

GSE successfully enabled the participating decision analyst to describe his own performed expertise as rule-instance traces. An example rule-instance was shown in Figure 6. (Note that at this stage in the GSE process, knowledge elements are descriptive labels only and opaque as to content.). A summary of the captured interview performance expertise is shown in Table 1.

Table 1. Summary Characteristics Of Decision Analyst's Self-Elicited Knowledge

SUMMARY CHARACTERISTICS	ELICITED INTERVIEWS		
	1	2	3
Recorded Interview Length..... (minutes:seconds)	51:43	42:44	38:27
IDENTIFICATION-LABELING EXERCISE 1:			
Number of Cues.....	49	62	43
Number of Interventions.....	92	76	70
Total Cues and Interventions..... (external, perceptual knowledge elements)	141	138	113
Number of Goal Shifts.....	6	9	6
IDENTIFICATION-LABELING EXERCISE 2:			
Number of Direction Changes.....	2	24	17
Number of Speed Changes.....	6	19	9
Number of Pace Changes..... (internal knowledge elements)	7	43	28

Two kinds of knowledge element and cognitive processing were identified by the decision analyst. The first identification-labeling exercise reported explicitly observable (on the ERTs), external perceptual knowledge elements used in the analyst's rule-based cognitive processing. The second exercise reported internal cognitive processing knowledge elements that were not directly observable on the ERTs. These

internal knowledge elements were recognized and reported by the analyst only after he "peeled-back" the external perceptual knowledge elements identified in the first exercise and focused his attention on the remaining cognitive processing activity. The analyst described this internal processing as management of interview pacing; a process in which an initial store of client "energy" (interest, willingness to participate, patience, etc.) is carefully spent by the analyst during the interview to achieve effective client probability assessment performance. The consequences of pace processing was the ongoing revision of the interview agenda operation in terms of agenda processing direction and speed.

During the association exercise, the decision analyst was presented with protocol listings that showed both external perceptual knowledge elements and internal pacing knowledge elements. Interestingly, the analyst associated both external and internal knowledge elements in the same rule conditions when composing the rule-instances that explained his interview performances (see Figure 6). The two concurrent thought processes were combined to direct the overall rule actions responsible for his task performances.

An attempt was made to determine the reasonableness of the decision analyst's self-elicited rule-instance trace descriptions of his interview performances. Harking back to [Miller, 1956], a simple type of granularity test was performed. If the analyst's identified knowledge elements correspond to Miller's information "chunks," then when used in rules they should satisfy the "seven plus or minus two" STM queue size constraint. This should be observable in the 238 rule-instances generated by the decision analyst.

The test involved counting the number of knowledge elements per associated rule-instance trace and collecting them into groups in ascending order. The result was graphed and compared against Miller's short-term memory capacity limitation of "seven plus or minus two" symbol labels [Miller, 1956]. The summary results for each interview is shown in Figure 7. Each bar represents the number of rule-instances containing a fixed number of knowledge elements in their left-hand-side. The horizontal axis plots the rule-instance bars in a rank order from 1 to 10. Associated conditions for all decision analyst rule-instances fall comfortably within the expected Miller maximum of nine. As such, analyst identification-labeled knowledge elements pass a necessary but by no means sufficient condition of reasonableness for equating them to Miller's information "chunks."

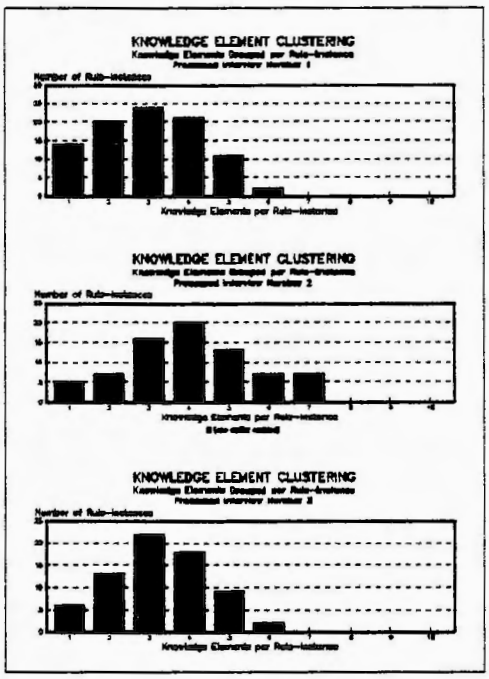


Figure 7. Decision Analyst's KE Clustering

Another test was developed to try and determine how well the self-elicited rule-instance traces actually reflected the decision analyst's impressions of his task performances. A measure of rule-instance complexity was concocted and applied to the interview rule-instance traces with the hope that rule-instance complexity might correlate with the analyst's retrospective impressions of the levels of expertise complexity he used in the different stages of the assessment interviews. For this purpose, the revised SRI/SDG agenda steps were used to represent different meaningful areas of expertise application.

The production system surface measurement of "condition elements per production" [Gupta and Forgy, 1983] was adapted to be an average measure of rule-instance complexity in each step of the analyst's agenda by restating it as the "average number of knowledge elements per rule-instance association." This measure was used to try to gage the expertise complexity used during each step of each interview. The assumption was made that the analyst's reported knowledge elements in all the interviews were of consistent granularity.

This complexity measure was used to analyze and compare the decision analyst's three interview performances as described by his self-elicited rule-instance traces. For each agenda step (total agenda and individual step) in each interview, the average number of knowledge elements per rule-instance was calculated. The interview performances were then ranked against

each other in an attempt to show the relative degree of complexity in the expertise used by the decision analyst. Measured total and step complexity was rank ordered from 1-3 according to measured increasing complexity. The comparative results are shown in Table 2. They closely matched the analyst's impressions of his interview performances.

Table 2. Relative Interview Expertise Complexity As Measured By Average Knowledge Elements Per Decision Analyst Associated Rule-Instance

BY AGENDA CONTEXT	ELICITED INTERVIEW NUMBER		
	1	2	3
Total Agenda.....	1	3	2
Goal: Motivation.....	3	2	1
Goal: Structuring.....	2	3	1
Goal: Conditioning.....	2	3	1
Goal: Pre-Encoding Practice.....	1	3	2
Goal: Encoding.....	1	3	2
Goal: Verification.....	1	2	3

The third step in the GSE methodology is to generalize the rule-instances created by the association exercise (see Figure 6). Such rule-instances are only lists of related knowledge element label descriptions. They disclose nothing of the information structured within. The generalization exercise was developed to enable an expert to fill out and elaborate the meaning and structure of his/her stimulus pattern perceptions.

The decision analyst was introduced to the O-V and O-A-V models for structuring information. He was asked to review his FLERTs and focus his attention on any and all interview performance factors that came to mind and seemed relevant to his thinking during the interview guiding process. From the first review of a single interview's expertise review tape, the analyst generated the list of 20 task performance related factors shown in Table 3.

The analyst's preliminary factor list showed only six factors where he naturally reported both a description and a value range. He did not however report specific factor values nor the knowledge elements containing them. The analyst expressed substantial uncertainty about the objective of the exercise and how to report. In subsequent discussion, a conceptual outline of the repertory grid approach [Hart, 1986] and feature analysis [Anderson, 1990] seemed to provide the analyst with the

Table 3. Preliminary Self-elicited Decision Analyst Performance Factors

1. Attitude	12. Degree of interest in process of elicitation (curiosity)
2. Experience	13. Attention to detail (picky vs sloppy)
3. Brightness	14. Degree of self-consciousness (ability to role-play)
4. Understanding of probability	15. Complexity of idea being presented (simple to complex)
5. Memory	16. Candor
6. Depth of procedural thinking about job that he does	17. Clarity of expression
7. Introspectiveness (intuitive vs analytical)	18. Length
8. Independence of thinking (independent vs job driven)	19. Ability to speculate (creative/free association thinking)
9. Ability to adjust thinking from heuristics (can they see the illusion)	20. Degree of connectedness with other intercepts (stand-alone vs part of single idea)
10. Prior preparation for elicitation process	
11. Breadth of view of job (tunnel vision on tasks)	

background metaphor he needed to more adequately address the exercise. With this guidance, the analyst reviewed his FLERT for interview 2 again. He first prepared a revised dictionary of personally significant factors with a list of possible values for each. Then he reviewed the tape again and recorded factors and factor-values for relevant knowledge elements on the formal protocol for the interview. An excerpt from his dictionary is the following:

Experience/familiarity with behavior of product
- 1) Significant experience but not recent -> fuzzy recall
- 2) Limited experience but recent
- 3) Significant recent experience
- 4) Significant recent experience and some personal analysis of the experience

Mental acuity - general
- 2) Average
- 3) Above average
- 3) Quick

Understanding of uncertainty/probability at intuitive level
1) No grasp of probability as a representation of uncertainty or variability
2) Limited grasp
3. Intuitive appreciation of function of probability

Perspective of job responsibilities and activities
1) Completely job-driven; has perspective of immediate manager or "job description" book; "company man"
2) Mixed
3) Independent/personally objective view of job responsibilities

Upon completion of this exercise, the decision analyst was asked whether his factors/factor values were local to only his identified knowledge elements or were generally present throughout the entire interview performance? He was unable to answer.

THE ABILITY TO DESCRIBE

Initial GSE research created a methodology enabling an expert to self-elicit his/her own performed knowledge as rule-instance traces of labeled information structures [Shephard, 1990a,1990b]. The labels are verbal names for a variety of possibly different kinds of cognitive processing units called, in GSE, knowledge elements. The decision analyst reported two separate kinds of concurrent cognitive processing that jointly participated in his rule-based cognitive processing. The first involved processing with external perceptual knowledge elements and the second processing with internally generated knowledge elements. Both kinds of knowledge element joined together in the common rule-based processing operation depicted in Figure 1.

While the GSE identification-labeling and association exercises provide appropriate guidance for producing a rule-instance "label" trace description of performed expertise (as described above), they leave undescribed the semantic content of the labeled knowledge elements involved. The semantic content of labeled knowledge elements is present in and/or cued by factors present in corresponding recorded ERT blocked frame sequences. The key to full, in-depth self-elicitation of performed expertise depends on an expert's ability to properly describe all the varied forms of knowledge and processing that may comprise his knowledge elements. What cannot be effectively described cannot be meaningfully reported.

To illustrate this contention, consider a parallel experience with the identification-labeling exercise described earlier. Two decision analysts actually participated in this research; the first in a pilot study and the second in the experimental interviews. The pilot study tested the feasibility of using

retrospective prompting to effectively rehearse prior perceptions and enable reporting of the perceived information structures (knowledge elements) used in ongoing cognitive information processing. When the pilot study analyst first viewed his own performed ERT, he was not given any detailed preparation or guidance. He was simply asked to identify and label passing videotape frame sequence blocks that he thought about as separate, meaningful entities when he led the pilot study assessment interview. He found he could not even begin; because of the following:

1. Lack of practice in considering his personal thought processes in rule-based terms

2. Lack of experience in consciously identifying and verbally labeling his recorded personal perceptions as distinct knowledge elements used in his own rule-based conscious cognitive processing.

To understand his difficulty, consider the following micro-scenario. The decision analyst is in the middle of an assessment interview. He is faced with a moving time-stream of perceptual sense data. Envision his visual and aural perceptions during one brief episode in his interviewing task performance:

> Following some action sequence by the analyst, the interview client reacts visually and aurally, through a brief time span. He frowns, looks away, coughs, pauses, shakes his head and looks again at the analyst. Taken together, in the overall context of the interview to this point, the analyst may take this time-spanning perception sequence to represent a meaningful information structure; perhaps a simple response interpreted to be a simple "yes" or "no." Perhaps something else. In the identification-labeling exercise the entire micro-episode must be categorized and labeled as a discrete, complete piece of structured information that is meaningful to the analyst's cognitive processing. It is a knowledge element he uses in his thinking process to guide his ongoing task performance. It is one of many knowledge elements he is asked to recognize, identify and label for the interview.

To enable the decision analyst to identification-label his own recorded perceptions, two forms of guidance were provided. First, the rule-based conscious cognitive processing model (shown in Figure 1) was discussed as a reasoning framework with the analyst. His understanding and acceptance of it re-oriented his perspective for relating to his own perceptual environment and interpreting his cognitive processing as "If... Then..." rule operation. Second, decomposition guidelines were formulated for the analyst to use in blocking his ERTs into tape

frame sequences that corresponded to his identification-labeled knowledge elements. The guidelines provided a syntax for decomposing the analyst's videotaped interview perceptions; a procedure for identifying appropriate beginning and ending points for labeled knowledge elements. Together, the framework model discussion and the decomposition guidelines enabled both the pilot study and experimental decision analysts to successfully complete the identification-labeling exercises they were assigned.

KNOWLEDGE ELEMENT REPRESENTATION

To get at the semantics of self-elicited expertise, the generalization exercise was developed. It asked the expert to accept the idea that his/her labeled knowledge elements contained important attributes or factors whose specific values determined the semantic role played by those knowledge elements in his/her thought processing. The O-A-V triplet was introduced to the decision analyst as a useful way to represent the information structures he had labeled as knowledge elements [Shephard, 1991].

The analyst attempted to "generalize" his knowledge elements by again viewing his FLERT. With that rehearsal, he developed a list of task performance considerations as they occurred to him (see Table 3). These considerations were expected to be the set of factor (attribute) descriptions needed to semantically complete his labeled knowledge elements. A brief dictionary of reported factor descriptions and their value ranges was prepared and factor labels plus values were added to the existing fully labeled expertise performance protocol. While of value in adding some further description of the ongoing rule-based processing, factor descriptions and values unfortunately did little to decompile labeled knowledge elements into understandable knowledge structures that truly explain knowledge element nature and purpose.

This realization prompted reexamination of the possible nature of the knowledge structures used by experts. Recent work by Slatter analyzed the knowledge engineering literature and identified five major classes of expertise models [Slatter, 1990]:

1. Heuristic models
2. Deep models
3. Implicit models
4. Competence models
5. Distributed models

He goes further to describe the knowledge categorization used in these models. Based on his discussion, it appears reasonable to summarize the knowledge types used in expert systems as follows:

Factual knowledge: widely shared domain knowledge available in textbooks and other public sources

Heuristic knowledge: rules of thumb or good practice compiled over time through experience in a specialist domain

Deep knowledge: variety of structures representing background knowledge of domain objects, hierarchical relationships between domain objects, causal models of how domain objects are related, specialist representations of domain objects, and how to reason from first principles

Implicit knowledge: nonarticulable experience-based knowledge that allows effortless, seemingly intuitive problem solving or task performance

GSE to date has concentrated on capturing a skeletal form of expert heuristic knowledge. Factual and deep knowledge available or cued in knowledge element tape frame sequences have not been effectively addressed. The nonarticulable nature of implicit knowledge would seem to put it beyond conscious reporting of the performing expert.

In addition to extending expert attention beyond heuristic knowledge to factual and deep knowledge, is the need to provide the expert with a more powerful structure for representing the forms of knowledge present in knowledge elements. A powerful structure for this purpose is the object. The idea of object representation for human thought began as a Greek innovation [Booch, 1991]. Descartes went on to observe in the seventeenth century that the object-oriented view of the world was a natural human approach [Stillings et al, 1987]. The nature and power of the object structure recommend it for comprehensive representation of all expert reportable knowledge within the framework of GSE.

AN OBJECT MODEL FOR KNOWLEDGE ELEMENTS

An object model for representing the variety of knowledge contained in labeled knowledge elements must be based on the GSE cognitive model (Figure 1) and the types of knowledge acknowledged to be used by performing experts. A single comprehensive object template is contemplated for all types of knowledge element. The reporting expert will use the appropriate subset of the object template's available representation capabilities to record the kinds of knowledge being reported. By describing the object model template as a checklist and worksheet, it can be used by a reporting expert to

comprehensively trigger, identify, analyze and record all the various types of knowledge present in each labeled knowledge element.

This approach would use the object template model to represent any and all knowledge elements - from those with simple to those with arbitrarily complex varieties of knowledge and processing in their composition. The cognitive information processing however, would remain simple and straight-forward as shown in Figure 1. It will be a case of simple rule processing with arbitrarily complex knowledge element symbols.

In its present form, the model is in two parts. The first is a knowledge element checklist worksheet intended for expert description of reportable attribute/values and cognitive processing. It provides for description of visual and aural feature patterns as recognized sensory attributes, mapping of sensory feature patterns into internal task considerations (factor/values and models) and sensory feature pattern mapping to recognized domain facts. Next, the worksheet asks the expert to report and describe his/her own initiative driving internal agenda processing and action-driving cognitive processing. The expert is asked to describe his/her deep knowledge in the form of quantitative and qualitative processing performed during the labeled knowledge element's task activity. As research results suggest that different cognitive processing streams may exist concurrently, the expert should address all forms of processing here. As a final item in the knowledge element description, the expert should record an explanation of the meaning and significance of the knowledge element - to the extent it has not been captured earlier.

The second part of the model is a formalization of the expert's reported description into a standardized presentation of knowledge element attributes and values, processing procedures that generate further attributes and values that contribute to the post-processing character of the knowledge elements and expert explanations of the knowledge element.

The status of current GSE research is planning and organizing new subjective probability assessment interview experiments. Using the same domain, the objective is to test the expanded descriptive power of GSE enhanced with the object model-based generalization exercises. If successful, it should provide a clearly useful comprehensive description of actual decision analyst expertise that can be used in either conventional approaches to expert systems development or in itself be used to "grow" the problem space necessary to develop an human emulation-based expert system.

REFERENCES

Anderson, J.R. (1990). *Cognitive Psychology - and Its Implications*. 3d ed. Freeman.

Booch, G. (1991). *Object Oriented Design with Applications*. Benjamin/Cummings, p. 35.

Bruner, J.S., Goodnow, J.J. And Austin, G.A. (1956). *A Study of Thinking*. Wiley.

Ericsson, K.A. and Simon, H.A. (1980). "Verbal Reports as Data," *Psychological Review*, 87, 3, 215-251.

Ericsson, K.A. and Simon, H.A. (1984). *Protocol Analysis - Verbal Reports as Data*. The MIT Press.

Gilmartin, A., Newell, A. and Simon, H.A. (1976). "A Program Modeling Short-term Memory Under Strategy Control." in C. N. Cofer ed. *The Structure of Human Memory*. San Francisco: Freeman.

Gupta, A. and Forgy, C.L. (1983). "Measurements on Production Systems." *Report No. CMU-CS-83-167, Department of Computer Science, Carnegie-Mellon University*.

Hart, A. (1986). *Knowledge Acquisition for Expert Systems*. McGraw-Hill.

Merkhofer, M.W. (1987). "Quantifying Judgmental Uncertainty: Methodology, Experiences, and Insights." *IEEE Transactions on Systems, Man, and Cybernetics, SMC-17*, 5, 741-752.

Miller, G.A. (1956). "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information." *Psychological Review*, 63, 2, 81-97.

Newell, A. (1972). "A Theoretical Exploration of Mechanisms for Coding the Stimulus." In A.W. Melton & E. Martin eds. *Coding Processes in Human Memory*, John Wiley, 373-434, 1972.

Newell, A. (1973). "Production Systems: Models of Control Structures." In W.G. Chase ed. *Visual Information Processing*, Academic Press, 463-526, 1973.

Newell, A. and Simon, H.A. (1972). *Human Problem solving*, Prentice-Hall.

Newell, A. and Simon, H.A. (1976). "Computer Science as Empirical Inquiry: Symbols and Search." *Communications of the ACM*, 19, 3, March.

Nisbett, R.E. and Wilson, T.D. (1977). "Telling More Than We Can Know: Verbal Reports on Mental Processes," *Psychological Review*, 84, 3, 231-259.

Shephard, G.G. (1990a). *Guided Self-Elicitation of Decision Analyst Expertise*. Dissertation: Arizona State University.

Shephard, G.G. (1990b). "Developing a Guided Self-Elicitation Methodology for Natural Human Expertise," *Proceedings of DSI 1990 Annual Meeting*, San Diego, CA.

Shephard, G.G. (1991). "Guided Expert Self-Elicitation: Generalizing Captured Expertise," *Proceedings of Hawaii International Conference on System Sciences (HICSS-24)*, Computer Society Press.

Slatter, P.E. (1990). "Models of Expertise in Knowledge Engineering", in H. Adeli ed. *Knowledge Engineering - Volume I Fundamentals*, McGraw-Hill, pp. 130-156.

Spetzler, C.S. and Stael von Holstein, C.A. (1975). "Probability Encoding in Decision Analysis." *Management Science*, 22, 3, 340-358.

Stael von Holstein, C.A.S. and Matheson, J.E. (1978). *A Manual for Encoding Probability Distributions*. prepared for Defense Advanced Research Projects Agency under subcontract for Decisions and Designs, Inc., SRI International, Menlo Park, CA, Final Report, SRI Project 7078.

Stillings, N., Feinstein, M., Garfield, J., Rissland, E., Rosenbaum, D., Weisler, S. and Baker-Ward, L. (1987). *Cognitive Science: An Introduction*. MIT Press, p. 305.

Tversky, A. and Kahneman, D. (1974). "Judgment Under Uncertainty: Heuristics and Biases," In D. Kahneman, P. Slovic & A. Tversky eds. *Judgment Under Uncertainty: Heuristics and Biases*, Cambridge University Press, 3-20, 1982.

Vandamme, F. (1987). "Knowledge Extraction from Experts in View of the Construction of Expert Systems", in J.L. Mumpower, L.D. Phillips, O. Renn and V.R.R. Uppuluri eds. *Expert Judgment and Expert Systems*, Proceedings of the NATO Advanced Research Workshop on Expert Judgment and Expert Systems held in Porto, Portugal, August 25-29, 1986, pp. 321-337, Springer-Verlag: Berlin Heidelberg New York.

Zechmeister, E.B. and Nyberg, S.E. (1982). *Human Memory*. Brooks/Cole Publishing (Wadsworth), Belmont, CA.

TERMINOLOGY AND KNOWLEDGE ACQUISITION:

EXPLORING A SYMBIOTIC RELATIONSHIP

Douglas Skuce
Dept. of Computer Science,
University of Ottawa, Ottawa, Canada K1N 6N5.
dsksl@csi.uottawa.ca

Ingrid Meyer
School of Translators and Interpreters,
University of Ottawa, Ottawa Canada K1N 6N5.
ixmal@acadvm1.uottawa.ca

ABSTRACT

The purpose of this paper is to describe ways in which the disciplines of terminology and knowledge acquisition can benefit each other. On the one hand, terminology (the discipline) involves a substantial knowledge acquisition component since it is centrally concerned with describing complex specialized concepts; on the other hand, knowledge acquisition requires correct and consistent terminology, since it is critically dependent on the efficient and unambiguous linguistic expression of concepts. Our discussion is based on experience with using a knowledge acquisition tool called *CODE* in two different terminology environments: terminology as a *distinct specialization*, and terminology as practised in *document-production*. In the first part of this paper, based on our experience, we discuss the knowledge acquisition needs that characterize each of these terminology-intensive environments; in the second part, we describe the terminological support that is currently available and under development in CODE, which is being used to build a prototype knowledge-based term bank called COGNITERM.

INTRODUCTION

Terminology[1] is the practical discipline[2] concerned with the *description* and *naming* of concepts in specialized fields. Since concepts are the starting point for all practical terminology work, this

---

[1] The term *terminology* is somewhat of an embarrassment to Terminology researchers because it is semantically "overloaded" (Cf. Section 1.2.1 of this paper), having three distinct meanings: 1) a scientific field of study (e.g. "Terminology has been influenced by fields as diverse as Philosophy, Psychology, Linguistics and Computer Science"); 2) the practical discipline of describing and naming concepts in specialized fields (e.g. "Terminology is practised in a variety of working environments"); and 3) the set of terms representing the system of concepts in a particular subject field (e.g. "The terminology of Artificial Intelligence is changing rapidly"). To reduce confusion, whenever it is stylistically possible, we use *terminology work* or *the discipline of terminology* to designate the second sense, which is the most commonly intended sense in this paper. To designate the first sense, we have capitalized the term.

[2] A description of the discipline of terminology is beyond the scope of this paper. Interested readers are referred to Rondeau 1984, Wüster 1974, and Sager 1990.

discipline clearly has an important knowledge acquisition (KA)[3] component: *describing* concepts involves acquiring knowledge about their *characteristics* or *properties* (a synonym), and *naming* concepts involves matching conceptual characteristics with linguistic forms. We believe that persons involved in terminologically-intensive activities would benefit from KA technology, and we have developed a KA tool called *CODE* (Skuce *et al.* 1989, Skuce 1991) which places a particular emphasis on providing support for terminology (Meyer in press, Skuce and Meyer 1990a). We have used the CODE system in two terminology-intensive applications (Meyer and Paradis 1991, Skuce and Meyer 1990b): one with the Secretary of State of Canada, and another with Bell Northern Research. Our work is concurrent with the increasingly prevalent view in the literature of Terminology (e.g. Ahmad *et al.* 1990, Auger 1989, Czap and Nedobity 1990, Knowles 1988, Parent 1989, Wijnands 1989) that repositories of terminological data (traditionally called *term banks*) are in fact evolving into knowledge bases in that they contain a large amount of *encyclopedic* knowledge in addition to strictly *linguistic* knowledge[4].

Our explorations into the benefits of KA to terminology have made it increasingly clear that the KA-terminology relationship is a *symbiotic* one: while KA technology can facilitate terminology work, support for terminology can also improve KA technology. In effect, terminology is just one aspect of the linguistic dimension of KA, which is being increasingly stressed in the KA literature (e.g. Barnett *et al.* 1990, Gaines and Shaw 1989[5], Shaw and Gaines 1989, Motta *et al.* 1990, Skuce and Monarch 1990). Hence, a major thrust of research on CODE is to develop terminological support facilities, and to integrate these with the other components of the system.

This paper is organized as follows: Part 1 examines the first side of the KA-terminology relationship, namely the need for KA support in the discipline of terminology. To this end, we analyse the KA problem in two contrasting situations in which terminology can be practised. Part 2 turns the tables to explore the other side of this symbiotic relationship, namely *the need for terminological support in KA technology*. In this context, we describe a number of existing and planned features for providing terminological support in CODE.

1. KNOWLEDGE ACQUISITION SUPPORT IN TERMINOLOGY

Terminology work can be carried out in a wide spectrum of working environments (Meyer and Skuce 1990). At one end of this spectrum is terminology practised as a distinct specialization. In this type of environment, which might be described as the most "pure" form of terminology, we find persons officially designated as terminologists, often with professional training and/or certification in terminology, following a controlled methodology. These *terminologists* (this is often an official job title) have the task of determining the essential conceptual and linguistic information about terms, and describing this information in the form of terminology records, for ultimate storage in a term bank[6]. At the other end of the spectrum one finds terminology practised

---

[3] By *knowledge acquisition*, we designate not only the *acquisition* (in the narrowest sense of this term) of knowledge, but also the various aspects of *managing* or *processing* this knowledge, e.g. *applying* it (as in typical expert systems, *critiquing* it (for debugging purposes), or *supplying* it (as in designers' assistants, teaching systems). As we mention in section 1.3, we prefer *knowledge management* as a generic term for all these activities.

[4] By *linguistic* knowledge, we mean the set of conceptual characteristics that are necessary and sufficient (in the mathematical sense) to define a term, while *encyclopedic* knowledge is the set of all other conceptual characteristics. For example, a necessary and sufficient definition of the term *river* might be something such as "a natural stream of water of considerable volume" (Webster's Ninth New Collegiate Dictionary). Information such as where rivers start and end, what kinds of fish they contain, their role in navigation, etc., would be encyclopedic. Cf. Meyer 1991 for more details.

[5] The articles by these authors contain some particularly interesting diagrams showing the types of terminological confusion encountered when dealing with experts.

[6] A term bank is a data base that stores terminological data by subject field. The largest term banks are all multilingual, since translators are the largest user category.

as a component of document-production. For example, it could be practised in an industrial setting, which is a situation we will examine later in this paper. This type of environment is characterized by a much more "casual" approach: the naming and description of concepts is carried out at various "links" in a "chain" of activities, which can include product design specification, technical writing (e.g. user manuals), revision, proofreading, translation, management information, and so on. Normally, the persons involved in these activities have no specialized training in terminology, their methodology is informal, and there is no centralized repository for the terminological data.

The spectrum of situations in which terminology work can be carried out dictates a range of KA needs. These are described in turn below, for each of the two "ends" of the spectrum.

1.1 Terminology as a distinct specialization

This environment is typified by organizations such as the Department of the Secretary of State of Canada, which has had an official terminology service since 1953. Here, the mission of terminologists is to facilitate the proper use of terms, in English and in French, throughout the Public Service of Canada. To this end, terminologists maintain what is now the largest term bank in the world (about one million database records). They also prepare bilingual glossaries (which are often published) on subject areas requested by clients, and respond to inquiries from clients on specific problems. The terminological data that are collected and provided include such things as definitions, synonyms, equivalents in the other official language, peculiarities of usage, etc.

The major challenge of terminology is conceptual, not linguistic: terminologists are trained in linguistics and thus are properly prepared for the linguistic dimension of their task; in contrast, they are not normally subject-field experts, yet they must acquire a substantial amount of expert knowledge in order to do their work. In this difficult task, they must often work alone, gleaning as much as they can from documents which they also must seek out. The major difficulty is in pinning down the *meanings* of terms, which are often used loosely, ambiguously, and inconsistently. Compounding their problem is the fact that terminologists can be required to work in several fields simultaneously, or to change fields frequently depending on clients' needs. Often a terminologist will be assigned to take up where someone else had left off, and much of what the first had done may not be easily accessible to the second.

In the following paragraphs, we analyze the four components of a terminologist's work in terms of the KA tasks on the one hand, and the roles of knowledge in the production of terminology records on the other.

1.1.1 Selection of documentation

KA tasks. Before any collection or analysis of terms can occur, terminologists must select the knowledge sources for the project. Given their linguistic orientation, they have traditionally preferred texts as knowledge sources, although the collaboration of experts is also highly valued. Before collecting the documentary corpus, terminologists first acquire some general knowledge about the field by doing introductory reading in textbooks, encyclopedia articles, popularizing journals, etc. Often, at this stage, terminologists will sketch out these "skeletal" knowledge structures in the form of a concept network, using paper and pencil. They will make lists of the most fundamental terms, and will make mental or written notes to clarify the corresponding concepts.

Roles of knowledge. These preliminary KA activities are crucial to the selection of the documentary corpus since they help to clarify the project's scope: a clear idea of the conceptual boundaries of the field helps delimit the range of documentation to be sought. Determining areas of overlap with other fields also helps terminologists establish links with related documentation. When the terminologists are ready to begin the detailed search for the documentary corpus, a clear

idea of the major subfields helps them orient their work along a number of documentary "paths", which may be prioritized according to users' needs. Having a general idea of the hierarchical structure of the field also helps orient the process of documentation selection since terminologists tend to proceed from general to more specific literature.

Once a preliminary corpus is obtained, their general knowledge of the subject-field provides terminologists with a yardstick for judging its quality. It also helps them classify documentation according to subfield. In multilingual terminology, which is becoming increasingly important, classification according to subfield is particularly important. In order to "manage" the large amounts of documentation to be *scanned* (see 1.1.2 below), terminologists very often work on one subfield at a time, in one language and then in the other, before proceeding to another subfield.

Finally, these preliminary activities provide terminologists with the conceptual framework and basic terminology needed for communicating with librarians and other documentation resource persons, as well as with experts. Communication is particularly important in the case of experts[7], who tend to be very busy; if terminologists have done their "homework", they will be able to direct the conversation in order to elicit the maximum information in a minimum amount of time.

1.1.2 Establishment of a nomenclature

KA tasks. Once the documentation has been selected, it undergoes a process called *scanning*, i.e. careful reading, with the extraction of potential terms along with their contexts[8]. Additional research may be needed for specific problems (e.g. terms not found for concepts identified, terms with inadequate contexts), after which the data is organized by grouping the various instances of a term, noting obvious cases of synonymy, polysemy, abbreviations, etc. During the scanning process, terminologists begin to analyze the general knowledge structures of the field, fleshing out (whether on paper or in the mind) the skeletal concept network drafted during their background reading. They also begin to analyze the conceptual characteristics of individual terms (i.e. the terms found in the documentation), based on the contexts in which the terms appear.

Roles of knowledge. Based on their general understanding of the subject-field, terminologists begin identifying the lexical items that are specific to their field. This process involves eliminating terms that would constitute "noise" in the terminology, i.e. lexical items that belong to general rather than specialized vocabulary, or terms that do not fall within the established boundaries of the field. As well, terminologists must identify lacunae in the preliminary terminology, i.e. areas of "conceptual space" for which no terms have yet been found. As terminologists prepare to finalize the nomenclature (i.e. to determine the terms for which records will be prepared) and decide which contexts will be retained for analysis (1.1.3), the conceptual framework acquired so far will help them continue to communicate about problem areas with documentation resource persons and subject-field experts.

1.1.3 Preparation of term records

KA tasks. Using the established terminology and associated contexts, terminologists can begin a systematic analysis of terms-in-context. The primary function of this analysis is to determine the meanings of the terms, although it also serves to identify other linguistic characteristics such as part of speech, gender, frequency, geographic origin, etc. The primary goal at this stage is to achieve the depth of understanding needed to complete the term records. Terminologists carefully analyze the various contexts in which a term has been found in order to identify a certain number of

---

[7] In effect, terminologists experience most of the problems of expertise elicitation summarized, for example, in Gaines 1990.

[8] Noting a variety of contexts for the terms is extremely important in terminology, since contextual analysis is the principal way in which terminologists determine the meanings of terms, and also how they are used in context. Most term banks include some contexts in the terminology records.

conceptual characteristics for these concepts. These characteristics will then be compared with those of potentially related concepts (e.g. synonyms, equivalents in the other language) in order to determine those which are necessary for establishing a conceptual match.

Roles of knowledge. The most important derivative of conceptual analysis is definition construction. If they are attempting to prepare the classic intensional (i.e. genus-differentia) definition, which is appropriate for many but not all concepts, terminologists will need to compare the characteristics of a given concept with those of concepts at the same hierarchical level (i.e. with the characteristics of the co-ordinate (sibling) concepts, in order to determine the distinguishing characteristics (i.e. the differentia in an intensional definition). Relations other than the generic-specific (e.g. whole-part, cause-effect, tool-function) may also be analyzed and reflected in definitions.

Second language terms. Conceptual analysis is also essential to identifying synonyms and equivalents in the second language. Identifying synonyms requires a careful comparison of conceptual characteristics in order to determine that these are indeed identical for the terms in question. When two concepts differ in only a very few (and not very significant) characteristics, they may be designated as pseudosynonyms (e.g. one concept may have one more characteristic than another, and thus be more specific). Establishing a conceptual match is also crucial to multilingual terminology work, which is complicated by the fact that conceptual structures often do not correspond perfectly from one language to another, resulting in cases of incomplete equivalence. Sometimes there may be no equivalent in the other language at all, resulting in the need to create a neologism[9]. In this case, conceptual analysis is essential for determining whether the concept already exists within the current knowledge structures underlying the target language, and when it does, what its characteristics are (since the concept is so new, its characteristics, and consequently its location within the knowledge structures, may still be fluctuating. In many cases, an existing term will be adopted to designate the new concept, and conceptual analysis of the candidate terms is essential for determining which one possesses the greatest semantic compatibility with the new concept.

1.1.4 Quality control

KA tasks. Quality control can be achieved by two types of activity: *revision* and *updating*. On the one hand, before the project is completed, the various types of information collected by the terminologist are revised by subject-field experts and other terminologists. On the other hand, after the project is completed, a periodic updating of terminology records can occur whenever this is justified by changes and expansion in the subject field. Revising the results of a terminology project involves analyzing and discussing specific conceptual problems identified by the experts and/or other terminologists. Periodic updating implies a monitoring of changes in knowledge structures and conceptual characteristics.

Roles of knowledge. To facilitate revision, terminologists need a certain depth of understanding of the subject field in order to interpret feedback from experts, and to elicit information on this feedback (e.g. when terminologists do not understand feedback, when the feedback contradicts what the terminologists believe, or when experts give conflicting feedback). In such interactions, the terminologist is functioning in many respects like the traditional knowledge engineer

[9] A neologism is a term that is used in a new sense. Sometimes existing terms can be used (and just given an additional sense), and sometimes a totally new term is coined.

1.2 Terminology as a Component of Document-Production

By *document-production*, we mean a "chain" of writing activities that are carried out from the inception of a product[10] to the production of public (or widely available) written information about this product. The "links" in a document-production chain can be distributed throughout an organization, the actual "documents" being in various states of completion: they can include anything from customer or management "wish lists", to product designers' rough personal notes, to intermediate "current state" documents used to coordinate members of a team, to "official" publications (e.g. technical manuals produced by technical writers), to translations of these official publications.

Although there are usually no officially designated terminologists in this type of environment, terminological activities are pervasive nonetheless (a fact that is often underappreciated by many, even those directly responsible for documents): concepts are described and named by persons such as product designers, technical writers, proofreaders, revisers, abstracters, management information specialists, public relations officers, and translators[11]. Given the heterogeneity of this type of environment, the terminology-related KA problems are much more complex than they are in the "purer" form of terminology work described above. We believe that many typical communication "foul-ups" can often be traced to inadequate care with terminology. The following are some of the issues that are relevant to alleviating such problems.

1.2.1 Choice of terminology

Given the variety of people involved in document-production, this kind of environment typically exhibits a lack of consistent methodology for terminology work, a situation often exacerbated by the pressure to meet deadlines. This problem is particularly crucial at the early stages of document-production. For example, the early contributers in the chain are the ones who usually introduce concepts and often choose terminology, but they generally have no formal training (and very often, no interest!) in terminology. Terms that are chosen "on the fly" easily become entrenched, even though they may be inappropriate. A typical resulting problem is *overloading*. "Let's just reuse this term that we all know, but give it a new sense "[12] In sum, this type of environment does not normally offer a discipline for assuring that terms are sensibly chosen, clearly described, and consistently used.

1.2.2 Coordination between links in the document-production chain

Given the number and type of people that can be involved in document-production, coordinating the various links in the chain is a fundamental problem. Writers in a given link in the chain may, for example, have trouble understanding what the originator of certain terms actually meant by them, and may have no simple way of finding out. The meanings of terms may have to be guessed or inferred from scant resources, somewhat as the terminologist working in isolation must do. Knowing that a given document will soon be passed on to another link in the chain, writers are easily tempted *not* to resolve terminological problems that they have inherited or are introducing, leading to a "Let the reader figure it out" phenomenon. A further complication is that documents do not flow in a one-way direction from inception to finalization. Consequently, writers can be sent in loops: A says it's X, but better ask B; B says it's Y, but better ask A. Common terminological problems resulting from this situation are *inconsistency* (terms being used to mean different things by different people), *overloading* (terms used in too many different senses), and

---

[10] For terminological simplicity, we use the term *product* to include things like services, regulations, committee decisions, etc. (and not just *products* in the strictest sense of the term).

[11] A more detailed overview of the various types of writing activities in a document-production chain can be found in *Language Technology* (April 1989, Special Issue on Documentation).

[12] A notorious example, beloved of computer specialists, is the term *object*. This term has so many senses in the computer literature that the only common denominator seems to be *thing*.

vagueness (no one can give a reasonable account of what a term means). This latter problem may (but not necessarily) be due to another problem that contributes to the coordination problem, namely that concepts exist at different levels of "clarity" at the various links in the chain: at the initial design stage, they may still be quite fuzzy; by the time they are documented in some kind of "official" text, their conceptual characteristics should be (in principle, at least!) much clearer. From a terminological point of view, this conceptual fluidity means a continuous evolution of concept definitions and names from one link in the document-production chain to the next.

1.2.3 Centralization of terminological data: from term banks to knowledge bases

Further complicating coordination between the various persons involved in terminological activities is the fact that most organizations do not maintain centralized, or even local, repositories (e.g. term banks) of terminological data. When such repositories do exist, they often take the form of informal glossaries that may be incomplete, out of date, not validated by experts and/or professional writers, and not used seriously throughout the group or organization. Often they only appear late in a project, when the requirement for formal documentation becomes unavoidable. Therefore, this state of affairs places a heavy onus on the writer to find out what the terms mean, who originated certain terms, whether they should be retained, how they should be used in context, how they should be translated, and so on. The lack of centralization of terminological data (particularly conceptual data) is particularly problematic for people who are at the *end* of the document-production chain, namely, the technical writers, editors, proofreaders, and translators[13]: they are the furthest away from the originators of concepts (and may have the hardest time accessing these originators); the documents passed onto them are likely to have the greatest number of terminological problems (due to the "Let the reader figure it out" phenomenon mentioned above); and finally, these people usually have the least amount of subject-field expertise (editors, proofreaders, translators, etc. are typically language experts, not subject-field experts). A lack of centralized information about terms and their meanings is also an obstacle for technical newcomers to a project, since it forces them to scrounge for knowledge about terms and concepts any way they can, usually by tying up existing personnel in long explanation sessions, another kind of inefficient "knowledge management" activity.

We feel the lesson should be clear: there is a serious need for a unified methodology by which linguistic and conceptual data would be maintained in a consistent, central, properly monitored knowledge base throughout the history of a project. all documentation, informal included, would be based on these data. Persons could still record knowledge in conventional forms (e.g. documents produced with word processors or "draw" programs), but the terminology would have to be consistent with the knowledge base[14]. More desirable, however, would be that documentation be created and maintained solely as an integral part of the knowledge base. We report on an initial experiment along these lines in section 1.3.2.

1.3 Applications of Knowledge Acquisition Technology to Terminology

The two types of situation discussed above represent PROBABLE ends of a wide spectrum of terminology-intensive activities. What all these activities have in common, however, is a great need for assistance in acquiring, formalizing, storing, refining - in other words, *managing* (the term we prefer) - a variety of knowledge about the concepts designated by specialized terms. To meet these needs, we have developed a general purpose *knowledge management* tool, called CODE

---

13 Terminological problems are such an impediment to translation, and particularly to machine translation, that some organizations are starting to impose an in-house "controlled language" on their technical writers. An interesting overview of this phenomenon can be found in Pogson 1988.

14 How this might be assured in practice is a difficult unsolved question to which we are giving considerable attention.

(Conceptually Oriented Design Environment; see Skuce *et al.* 1989; Skuce 1991). CODE has now received preliminary testing in each of the two types of terminology-intensive environments described above[15]. Based on what was learned in these applications, we are now further enhancing the system for terminology applications (2.2, 2.3). In the next two sections, we briefly summarize the two applications of CODE with which we have had experience, and which we believe represent the two "ends" of the terminology spectrum.

1.3.1 CODE as a terminologist's assistant: the COGNITERM project

In the fall of 1989, CODE was tested in a pilot research project by the Terminology and Linguistic Services Directorate at the Department of the Secretary of State of Canada. Terminologists are keenly aware of the importance of conceptual analysis in their work, and the goal of this project was to investigate the possibility of eventually integrating CODE into a terminologist's workstation, which is currently in the early design stages at the Secretary of State. Specifically, the Secretary of State was interested in observing how useful CODE would be in helping terminologists acquire, systematize and retrieve the specialized knowledge needed to prepare accurate terminological records.

During the pilot project, CODE was used to assist in the preparation of a published bilingual vocabulary for the subject-field of the graphic arts. Specifically, we focussed on the subfield of typesetting, which presented a common conceptual challenge in that it has been evolving rapidly with computerization, thus presenting the problem of clarifying emerging concepts. Since an outdated vocabulary for the graphic arts already existed, we also wanted to see how useful CODE would be for revising previously drafted definitions, i.e. for detecting errors and inconsistencies. The project involved, as a principal knowledge source, a Secretary of State terminologist who specialized in the graphic arts. Subject-field experts were also consulted, and a certain amount of research was done by a visiting terminologist who had no previous knowledge of typesetting or experience with CODE. The knowledge provided by these persons was entered into the system by an experienced CODE user, assisted by the Secretary of State terminologist.

The methodology was consistent with that usually followed by terminologists at the Secretary of State:

1. Based on introductory reading and the knowledge of the Secretary of State terminologist, the subfield was delimited and a preliminary partitioning established. During this stage, CODE was used to provide a conceptual "sketch" of the field. Only a skeleton knowledge base was created, with the focus being on the principal properties of concepts.

2. Based on this preliminary work, a more detailed documentary search was carried out. It was found that the documentary search was facilitated by the delimitation and partitioning of the field established in step 1.

3. Based on the additional knowledge provided by step 2, and on periodic consultation with subject-field experts, the knowledge base was fleshed out. We essentially proceeded by filling out one sub-tree at a time, using mainly a top-down approach complemented by some bottom-up work. It was part way through this stage that the visiting terminologist joined the project. He found that the partially built knowledge base, because of its formalization and CODE's user-friendly, hyper-text interface, helped initiate him to the subject field in a surprisingly short time.

4. The visiting terminologist spent some time drafting definitions based on the information that had been entered into the knowledge base. He found the task to be greatly facilitated by the graphical

---

[15] The present paper only summarizes these experiences. The interested reader is referred to Meyer 1991, Meyer and Paradis 1991, Meyer and Skuce 1990b for additional information.

display, which clearly indicated the genus term (i.e. generic concept), and by the ease with which the conceptual descriptions of coordinate concepts (concepts at the same hierarchical level) could be compared in order to determine the differentiating properties.

As reported in the official journal of the Secretary of State (Meyer and Paradis 1991), the results of this application of CODE were extremely encouraging, and the Secretary of State has decided to retain support for conceptual analysis as a desiderata for its terminologist's workstation. A knowledge base of 95 concepts was created after fewer than 40 person-days of work. Of these concepts, 72 were completed to the level of detail judged desirable by the Secretary of State terminologist, while the remaining 23 were rather "skeletal", and were included only to illustrate the place of typesetting in the graphic arts (i.e. they were not typesetting concepts *per se*, but other high-level graphic arts concepts). The knowledge base was found to be useful in revising previously drafted definitions, and in drafting definitions for new concepts. Finally, the terminologists enjoyed working with the system and expressed an enthusiastic willingness to participate in any future CODE projects.

The following specific advantages of CODE were noted, both orally and in the end-of-project reports (Paradis 1990; Meyer and Paradis 1991):

1. The ease of entering, retrieving and changing information.

2. The usefulness of the graphical display for giving an overview of the subfield and for helping to maintain conceptual clarity.

3. The usefulness of inheritance mechanisms for eliminating manual repetition of information at lower conceptual levels, and of associated mechanisms for detecting inconsistencies.

4. The compatibility of the CODE methodology with that normally used by terminologists. This resulted in a short learning curve; the two terminologists became quite comfortable with the CODE methodology within two weeks.

5. The usefulness of the system as a training tool. As previously mentioned, the visiting terminologist, who had no experience in the field of typesetting, was able to learn the basic concepts very quickly due to the formalized knowledge base and hypertext-like system interface.

6. The usefulness of even a partially completed knowledge base for orienting documentary research and consultation with subject-field experts, as well as for placing emerging concepts.

It was also suggested that a CODE knowledge base, given its formalized structure and developed graphical capabilities, could help the work of standardizing committees[16] by providing a useful common basis for discussion. It was also proposed that the CODE system could be useful for various types of teamwork, since the formalized structure of the concept descriptions coupled with the system's hypertext-like interface make the knowledge base easily shareable.

Encouraged by these positive results, we have now launched a 3-year research project, called the COGNITERM project, which has three goals: 1) to further develop the CODE system with a view to specializing it for eventual use in a terminologist's workstation environment; 2) to develop a methodology for carrying out computer-assisted conceptual analysis in terminology; and 3) to build a prototype term bank (COGNITERM) using the system and methodology. The distinguishing

---

[16] Standardizing committees are responsible for discussing particular terminological problems. A common example is the selection of a preferred term when several competing terms exist for a given concept. Conceptual analysis is a very important component of the work of standardizing committees, who must be clear on the concepts in question before making decisions on which terms should be associated with the concepts.

feature of the COGNITERM term bank is that it features a rich and formally structured knowledge component. In short, it could be described as a hybrid between a term bank and a knowledge base, i.e. as a *knowledge-based term bank*. At the moment, we are focussing on the subject-field of optical storage technologies. Figure 1 shows a *concept descriptor* (cd), CODE's main frame-like knowledge unit, for the concept 'CD-ROM', as well as a graphical display of a part of the conceptual network in which this concept appears. In the cd, the "Conceptual Information" zone contains a traditional knowledge-based component (i.e. list) of the concept's characteristics (some of which inherited from more generic concepts, others which were introduced for this concept specifically). The "Linguistic Information" zone contains the information that is found in conventional term banks, e.g. synonyms, French equivalents, collocations. Finally, the "Definition" zone contains a variety of definitions (in this case, only an intensional definition is specified), which we see as a bridge between conceptual and terminological information.

1.3.2 Using CODE to support technical documentation

The Advanced Services Division of Bell Northern Research has developed a prototype design workbench called *Telos*, a highly graphic, object-oriented system for telephony system designers. The CODE system was used to develop a description of the basic concepts in Telos, which amounted to approximately 270 cds. The process involved two knowledge engineers experienced with CODE and approximately ten Telos experts entering concept descriptions and discussing terminology. Over about a year, hundreds of person hours were required. The use of CODE was not introduced until the project was well advanced (a situation over which we had no control); however no documentation activities had yet begun, and very little existing documentation was available. This was both an advantage, in that a considerable degree of freedom remained for rationalizing terminology, and a disadvantage, since the knowledge base had to be built largely by direct expert interviews. The result was a complex knowledge base which is nevertheless quite readable as a printed document in the natural language format that CODE provides. It clearly defines the conceptual structure, terminology, properties, constraints, assumptions, etc. of the system in a manner far superior to any documents that existed at that time. It later was used by documentation personnel to produce user reference manuals by exporting the cds into a publishing system. The documentation personnel were very enthusiastic about using CODE, and hoped that it might become a company-wide norm some day.

Our methodology for building the knowledge base was as follows. Telos designers would interact with the knowledge engineer, usually one-on-one, in intensive knowledge acquisition sessions using CODE. During these sessions, they would map out the concepts and properties of some part of the system, and enter preliminary descriptions into CODE until the expert thought that his/her ideas were correctly represented. About once a week, there would be "group reviews" in front of CODE (a high resolution projector would have helped!) involving the senior design team members (usually four), in which the concepts not yet clarified were discussed in often lively exchanges. By the use of pseudo properties attached to concepts such as "knowledge enterer", "date of entry", and "status of review", CODE aided in the decision making and knowledge debugging by recording the status of cds or properties as they were debugged and by permitting rapid review of the current structure. CODE's search/rename browser (see section 2.2) was frequently used to locate all occurrences of a term, either to clarify how it was being used, or to rename it. In this experiment, the eventual preferred method of use of the CODE system was to have the software engineers (the Telos designers), after they had been exposed to the system for a six-month period, including a three-day tutorial, enter their ideas into the system in a rough draft (brainstorming) form, not bothering to get it "too right". Then, a knowledge engineer would attempt to polish it, checking with the originator. Next, group meetings were held in which the other Telos engineers critiqued the *content*, refining and adding to it, while the knowledge engineer critiqued the *form* and *methodology*. Finally, the documentors would inherit the cds, and often would find additional problems that they were responsible for resolving.

We believe the use of such a tool permitted a degree of clarity, an economy of expression, and a facility for critiquing and debugging that could not have been provided by unstructured natural language and word processors.

Figure 2 shows a typical view of CODE in this application. In the *browser* (the main editing/navigating tool) at right, the portion of the knowledge base above and below *specification* has been shown, and the cd for *actor specification* has been selected. We can see, for example, that *behaviour ports* is an optional property introduced for the concept 'actor specification', i.e. it has not inherited from above, and hence actor specifications are the most general thing having behaviour ports. All property names and bodies were written in a constrained English syntax (Skuce 1991b) and all terminology can immediately be identified as referring to: a) a concept, b) a property of some concept, or c) other. If desired, the system can check the syntax and lexical status of each term, phrase, or statement. To the left of the property browser in figure 2, we see part of the whole cd hierarchy graph, which is several screens in size. The graph is covered by part of the cd for actor specification, which is displayed here in its full (i.e. not natural language) format. The concept names that appear within angular brackets (< >) show where the property has inherited from.

Conclusions: The following general observations and conclusions were made from this experiment:

1. It is essential to have someone familiar with the problems of conceptual analysis to assist in launching and overseeing the knowledge acquisition process. It is not realistic to assume that knowledge of this complexity can be acquired reliably by untrained users entering their ideas directly into CODE or, we believe, any KA system at present. There is simply too much experience required on matters such as how to structure the knowledge base (most importantly at the top levels), how to choose terminology, how to express logical constraints, etc. (Kitto 1989 has similar comments, and notes problems with terminology.) Software engineers are not trained in subjects such as linguistics, logic, and terminology, although perhaps some day they should be.

2. If the CODE system had been used from the outset of the Telos system design, we would expect that we could have avoided many of the problems we encountered, both in trying to clarify the conceptual structure and attempting to discard entrenched yet undesirable terminological choices. Presumably many delays in development would have been shortened or eliminated as a result of the improved clarity of communication (i.e. less time spent wondering and arguing about what was meant by a particular term, or working with an incorrect assumption about a meaning). Although we have considered for some time how this might be objectively and convincingly demonstrated, we have not yet thought of a way.

3. The resulting knowledge base serves at least the following purposes: a) it assured the designers that they had a unified view of the concepts and terminology; b) it was used to drive the production of training, reference, and maintenance documentation; c) it was useful to assist newcomers to more quickly "come up to speed" on the project.

4. Clearly, it would be desirable for CODE to automatically take on as many of the functions of the knowledge engineer (and other support personnel such as documentors) as possible, most importantly, as a critic. The present system takes no "initiative", except to report apparent contradictions due to properties inheriting with conflicting values. Often, the knowledge engineer's function is to prompt the expert when he appears uncertain about what to say next, and to critique knowledge as it offered (e.g. "I don't think the term "object" should be given any more meanings! Can we think of a more selective term?"). We do not believe that the kind of knowledge captured in the Telos experiment can be more than minimally critiqued by any automatic system forseeable in the near future (Shadbolt & Wielinga 1990 discuss such possibilities). There is simply too much world, technical, linguistic, logical, and other kinds of knowledge needed; indeed, probably few

knowledge engineers have all the needed skills. Nonetheless, some forms of semantic checking are possible, similar to a good compiler, and more of these will be incorporated into the next version of CODE (see section 2.3).

The only reported project similar to our Telos experiment of which we are aware is that reported in Devanbu *et al.* 1991. Here, the Kandor system (a KL-ONE family member) was used to reverse engineer a system developed by AT&T. About 200 frames (equivalent to cds) were created, but the methodology is not discussed in detail. No particular mention is made of the importance of terminology or features designed to assist with it.

2. TERMINOLOGICAL SUPPORT IN KNOWLEDGE ACQUISITION TECHNOLOGY

In the first part of this paper, we argued that there is a substantial KA component to terminological activities. In this section, we "turn the tables", as it were, to propose a number of features that KA tools can make available to assist users with problems of a terminological nature. Unfortunately, very few existing KA publications have shown any concern for terminology: a notable exception is the work of Gaines and Shaw 1989 and Shaw and Gaines 1989, which directly describes the importance of terminology. Some existing tools that provide limited support for terminology are reviewed in 2.1. In 2.2 and 2.3, we describe the terminological support in the present and forthcoming versions of CODE. Finally, in 2.4, we outline ideas on terminological support that might be envisioned for KA technology in the longer term .

2.1 Support for terminology in some existing KA tools

Relatively few KA tools that have been reported in the literature offer any direct support for terminology *per se*. However, a few have provided support for scanning texts, which are either existing documents or transcripts of expert interviews, with a view toward extracting not only terminology but hopefully all relevant facts from which to assemble a knowledge base. They typically allow the user to construct an arbitrary network that interrelates text fragments however the user wishes. The following are typical:

Acquist (Motta *et al.* 1990) is a component of the KEAT-2 KA tool. Using it, a knowledge engineer can scan a document and locate all relevant terms, phrases, or large sections of text. The terms can be entered in various lexicons, and they or other text elements can be linked in an informal network permitting rapid navigation of the document for further knowledge "mining". The system includes a standard set of generally useful links, but the user can create new ones.

Kriton (Linster 1989) is a KA tool incorporating a text analysis component similar to Acquist. It has a dictionary permitting it to recognize nouns, and the user can construct hierarchies from these, although there does not appear to be an actual frame structure (with property inheritance) underlying the hierarchies.

KADS is a well known KA methodology, described in Hickman *et al* 1989. Unfortunately, the function of a lexicon is barely mentioned in this report (four lines on p. 82); it is considered to be merely a list of words that could be a "useful guide". No consideration is given of how to construct it or what it should contain. (We consider this a good example of the lack of appreciation for the importance of terminology in the KA literature). KPT, a system based on KADS (Anjewierden 1988), again permits hypertext-like structuring of arbitrary text, which again normally consists of expert interviews.

Cognosys (Woodward 1990) permits a structured collection of text fragments to be assembled, but the text is normally prepared for each Cognosys component, i.e. it must suit the purposes

Cognosys requires. The knowledge engineer may select specific terms in the text as having various kinds of significance.

2.2 Support for terminology in the current version of CODE

We do not have space to describe CODE in detail (see Skuce *et al.* 1989; Skuce 1991), but we outline the general features specific to the needs discussed above.

User interface. The interface elements are designed to support the entire spectrum of terminology environments, which means that they must be usable by persons with various levels of linguistic and subject-field expertise. For example, one permits a pseudo natural language output to be generated (figure 3) for easy reading by experts. We thus emphasize the importance of CODE as a communication tool, for example as an interface between links in the document-production chain, as a basis for discussions between terminologists and experts, and so on. Hence the displays are intended be easy for domain experts to interpret.

Graphical display. Our experience has shown that graphical displays of knowledge structures greatly enhance conceptual clarity. CODE has several graphing options, both for hierarchies and arbitrary networks of concepts related by their properties. Both graphic and text displays have a boolean "filtering" facility which can be used, for example, for a) focussing on certain parts of a conceptual structure, b) isolating "special" (i.e. unclear, unconfirmed) concepts, c) showing, comparing, and contrasting multiple concept structures, and d) showing various hierarchical and non-hierarchical relations.

Quality control. Quality control mechanisms are essential for ensuring conceptual and linguistic consistency. This is particularly important when many persons are contributing to the construction of the knowledge base (e.g. in the document-production chain). CODE has a facility for consistency-checking based on inheritance mechanisms and features for "flagging" and locating concepts in varying degrees of finalization.

A global lexicon system. Apart from the normal cd knowledge base, the user may create a set of lexicons arranged in a tree structure. In certain situations, CODE may be used to assist in checking terminology in normal documents. Any term in a document not found in one lexicon is sought in its parent lexicon. Higher-level lexicons are intended to contain more general terms. Thus the "top level" lexicon could be (we haven't done this yet, see 2.3 below) a complete dictionary of general English, while lower-level ones would include only lexical items that belong to a specialized field of knowledge, or that are specific to an organization. Lexicon entries include information on multiple senses, parts-of-speech, informal definitions, comments, and pointers into the actual knowledge base, i.e. for a term that is a full-fledged concept unit in the knowledge base, one may consult the cd for clarification of its meaning.

Term extraction/checking from text.[17] Users may scan an existing text for its terminological content, for various purposes: a) to create a list of the important terms, b) to create cds for the important terms, c) to check on the usage of known terms. In the first two cases, when it detects a term not in the lexicon, the system stops and requests that users make an entry for it if they wish. In the third, the scanner may be set to stop on any terms known to be problematic, so that users (typically editors) may verify that their use is correct. Suitable alternatives are then made available at a menu click, as in a typical spelling checker. If they need help in understanding the term, users may decide which is the best alternative by exploring the use of the term in the knowledge base. As an example, suppose 'memory block' is a concept in the knowledge base

[17] A number of reports of systems intended to extract terminology and/or knowledge directly from texts have been reported, e.g. Moller 1988, Slator 1989, Jacobs and Zernik 1988, Moulin and Rousseau 1990, Reimer 1990, David *et al.* 1991, Szpakowicz 1990.

and has a property called 'size', the preferred term, meaning 'number of bytes'. (Confusion may arise between the distinction between "number of elements" and "number of bytes"). If it is stated in the knowledge base that people may refer to 'size' as 'length', or confuse the two meanings, all occurrences of 'length' or 'size' in the document can be checked to see if any refer to memory blocks, and to disambiguate them.

Simple English syntax and parser. While not directly a "terminology" feature, we have included in CODE a simplified, "English-like" syntax called ClearTalk (Skuce 1991b), which is available if the user wishes. Any noun phrase or statement can be checked for terminology or correct syntax, since many terminological problems are context-dependent. (Noun phrases are the norm for concepts and properties). An easily interpreted parse tree and diagnostics are presented to help the user correct syntactic or terminological errors. For example, if the user entered "the hinge of a fork", the system would say, "according to my knowledge, the concept 'fork' does not have a property called 'hinge'", and it would then display the cd for 'fork'. The user could then, e.g., decide either that a) he did not mean 'fork', or b) he did not mean 'hinge', or c) add to the knowledge base the fact that for him, forks have hinges.

Search/rename browser. This feature permits scanning the entire knowledge base for every occurrence of a term. It can be restricted to certain contexts to speed the search. A menu of all occurrences is presented, and the user may examine any one by a click. Occurrences of terms that are incorrect can be then be flagged by the user, to be replaced by a more appropriate term. We have frequently used this to resolve uncertainties about a choice of a term.

Property naming assistance. When choosing the name of a new property (a common terminological problem in knowledge-base building), the system can display all currently used names of related properties. Optionally, the user can immediately see the complete text of each occurrence of a selected property, to clarify what it means. Often, either a) the property should be generalized to some superconcept (CODE helps with this), b) the name choice is inappropriate, or c) it *does* make sense to use the same property name in two senses (e.g. 'length' in the sense of time or 'length' in the sense of distance). In addition, for users choosing to follow the the CODE syntax rules, property names can be parsed for conformance (they must be either simple noun or verb phrases).

2.3 Support for terminology in the forthcoming version of CODE

We plan the following extensions to the above features in CODE version 4[18], expected to be operational in late 1991:

Ability to use the same term for a concept in several senses. While the existing version has some support for property names used in multiple senses, it does not permit different concepts to have the same name in a given knowledge base. Version 4 of CODE will have a notion of *context*[19], and senses will be relative to a context. There will be a complete decoupling of naming and internal reference in the knowledge base, a serious limitation in the current version. A term may have multiple contexts. To use a standard example from the Linguistics literature, there could be two contexts, 'banking' and 'rivers'. In a 'river' context, the term *bank* would mean 'river bank', for example. To refer to this concept from the 'banking' context, one would say, e.g., 'the bank (of a river)'.

Ability to substitute a term with a quasi-synonym. It will be possible to replace a term with a quasi-synonym which is appropriate within a context or at a level of detail. For example,

---

[18] Version 3 was a prototype for Version 4, written in Prolog. Version 4, like Version 2, is written in Smalltalk.

[19] Our notion of *context* involves in part the linguistic notion of *homonymy* (the presence of several unrelated senses for a given lexical item).

the property 'spouse' for 'married person' should be called 'wife' in the concept 'husband' and all its subconcepts, but it is still the same property. (This is related to the coupling problem described above, and is solved via the same mechanisms.)

Ability to suggest possible synonyms or quasi-synonyms by accessing a thesaurus. This feature would be similar to existing thesaurus tools available for word processors, except that it would be integrated into the knowledge base, i.e., definitions would be actual knowledge base concept units. The user would ask, for example, "show me concepts with similar meanings".

Semi-automated definition generation. We are investigating techniques (deriving from Maybury 1990) to automatically generate, using heuristic algorithms, a possible definition for a term based on *concept similarity* on the one hand (e.g. similarity of characteristics of co-ordinate concepts) and *prototypicality* on the other (e.g. the degree to which a concept's properties are reflected in its subordinate concepts).

A clearly defined set of terminological "status levels". By *status level*, we mean attributes of a term such as how it is used (e.g. as a concept name or a property name), whether it is defined or not, whether it is used in definitions but is not a knowledge base concept or property, whether it is a term used only in casual comments, whether it is a term not yet in the knowledge base or a term that should not appear in the knowledge base, and so on.

The design of version 4 is based on the fundamental premise that *the knowledge base and its associated linguistic tools and rules should be seen as an integrated whole, not as two separately developed, loosely coupled modules.*

2.4 Toward more complete natural language support for knowledge acquisition

The following are a number of facilities that we feel would improve terminological support in KA technology for the longer term, i.e. over the next decade or so:

Incorporation of a full commercial English dictionary and thesaurus. One obvious step would be to incorporate a full dictionary and thesaurus of English, analogous to what is now available for word processors. Querying such a resource would provide the same type of assistance that it does in its present, unintegrated form (i.e. by looking in separate dictionaries or thesauri), except that ultimately one would want the dictionary structure to look the same as the knowledge base structure, so that the dictionary entries would be just a special kind of knowledge base units.

Expert terminological critiquing. This capability would be an extension to that provided by so-called "style checkers" that exist today for word processors. Instead of being developed independently of the knowledge base however, as in normal style checkers, critiquing rules should be developed as an integral part of the associated knowledge base (Our 'length' example above illustrates this idea; it is knowledge about the possible misuse of the term). Terminological critiquing can be envisioned in two distinct situations: 1) a "checker" for existing full natural language texts, such as technical documents being edited, or 2) a "monitor" for knowledge being entered into a knowledge base system such as CODE. (Both of these can be seen as extensions of the features described above.) The system would maintain rules (created by skilled editors) of typical errors in lexical choice, such as overloaded terms, improper synonyms, or incorrect senses. Upon encountering such a term, the system would attempt to decide the possible sense(s), display them to the user, and suggest either 1) replacing the term with a less ambiguous one, or 2) somehow disambiguating it, e.g., by a footnote or subscript.

Additional knowledge. The following are just some examples of requests or questions that a user might seek answers to in either a KA system or a text critiquer[20]:

show me an example or picture of this
what is the French equivalent of this term?
show me possible synonyms, antonyms, etc.
show proper usage, e.g. what verb to use with this noun
show me functionally related terms, e.g. "what to you call something that integrates?" ("an integrator")

A notion of "viewpoint". A given concept may be "seen" from various "viewpoints", which correspond roughly to the needs or interests of the writer or reader. For an example, various users might require a variety of knowledge about a certain laboratory procedure. There would need to be mechanisms to allow one user to say "show me only things about this laboratory procedure related to the tools that are required", and another to say "show me only things related to the types of organisms the procedure can identify". The notion of viewpoint can be extended to include a notion of depth of subject-field expertise. For example, the user may request information about the laboratory procedure that would be of interest to (and understandable by) a beginning biology student, or to a seasoned researcher.

Integration of true natural language understanding. Clearly, an ultimate target is the integration of full natural language understanding capabilities into knowledge management systems. To date, we are aware of only one project that is beginning to consider this: the CYC project (Lenat *et al.* 1990, Barnett *et al.* 1990).

ACKNOWLEDGEMENTS

Development of the CODE KA system and associated methodology (Skuce) has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the University Research Incentives Fund (URIF) of the Government of Ontario, Bell Northern Research, and Research Services of the University of Ottawa. The COGNITERM project (Meyer and Skuce) is supported by the Social Sciences and Humanities Research Council of Canada (SSHRC). Testing of the CODE system in terminology-intensive environments has been supported by the Terminology and Linguistic Services Directorate of the Department of the Secretary of State of Canada, and by Bell Northern Research.

REFERENCES

Ahmad, Khurshid, Picht, Heribert, Rogers, Margaret, and Thomas, Patricia. (1989). "Terminology and Knowledge Engineering: A Symbiotic Relationship Explained". *Technical Report*, KITES Project, University of Surrey, Guildford, Surrey.

Anjewierden, A. (1988). The KADS System. In: *Proc. First European Workshop on Knowledge Acquisition for Knowledge-based Systems*, (Addis, T., Boose, J., Gaines, B., Eds.) Reading, UK.

Auger, Pierre. (1989). "Informatique et terminologie: revue des technologies nouvelles." *META*, 34(3).

Barnett, Jim, Knight, Kevin, Mani, Inderjeet, & Rich, Elaine. (1990). "Knowledge and Natural Language Processing", *Communications of the ACM* 33, pp. 50-71.

Czap, Hans and Nedobity, Wolfgang (eds). (1990). *TKE '90: Terminology and Knowledge Engineering* (2 vols). Frankfurt: INDEKS Verlag. (Proceedings of the 2nd International Congress on Terminology and Knowledge Engineering, Trier, Oct. 1990)

---

20 The latter three types of question would require precise semantic relations to be defined. We have in mind basing these on the work of Mel'cuk (1988), who has developed a model of the lexicon (i.e. the Explanatory Combinatorial Dictionary) that stresses lexical relations.

David, Sylvie & Plante, Pierre. (1991). "Le progiciel TERMINO: de la nécessité d'une analyse morphosyntaxique pour le dépouillement terminologique des textes." *Colloque sur les industries de la langue*. Québec: Office de la langue française, pp 71-88.

Devanbu, P., Brachman, R., Selfridge, P. & Ballard, B. (1991). Lassie: A knowledge-based software information system. *Comm. ACM*, 34(5), pp. 35-49.

Gaines, Brian. (1990). "Knowledge Acquisition Systems," *Knowledge Engineering (Vol. 1: Fundamentals)*, Hojjat Adeli, Ed., New York: McGraw-Hill, pp. 52-99.

Gaines, Brian and Shaw, Milred. (1989). "Comparing the Conceptual Systems of Experts". *Proceedings of the 11th International Joint Conference on Artificial Intelligence*,Los Angeles, pp. 633-638.

Hickman, F., Killin, J., Land, L., Mulhall, T., Porter, D. & Taylor, R. (eds). (1989). *Analysis for knowledge-based systems. A Practical Guide to the KADS Methodology*. Ellis Horwood, Chichester.

Jacobs, Paul and Zernik, Uri. (1988). "Acquiring Lexical Knowledge from Text". *Proceedings of the AAAI*, pp. 739-744.

Kitto, C. (1988). "Progress in automating knowledge acquisition tools: How close are we to replacing the knowledge engineer?" In: Boose, J., and Gaines, B. (eds) *Proceedings of the Knowledge Acquisition for Knowledge based Systems Workshop*, Banff.

Knowles, Francis. (1988). "Lexicography and Terminography: A rapprochement?" *Zürilex '86 Proceedings*. Ed. Mary Snell-Hornby. Tübingen: Francke Publishers.

Lenat, Douglas, Guha, Ramanathan, Pittman, Karen, Pratt, Dexter, and Shepard, Mary. (1990). "CYC: Toward Programs with Common Sense". *Communications of the ACM* 33, pp. 30-49.

Linster, M. (1989) Towards a Second Generation Knowledge Acquisition Tool. *Knowledge Acquisition* 1, pp. 163-183.

Maybury, Mark T. (1990). "Generating Natural Language Definitions from Classification Hierarchies". *Proceedings of the 1st ASIS SIG/CR Classification Research Workshop*, Toronto, pp. 101-108.

Mel'cuk, Igor. (1988). "Semantic Description of Lexical Units in an Explanatory Combinatorial Dictionary: Basic Principles and Heuristic Criteria". *International Journal of Lexicography* 1, 3.

Meyer, Ingrid. (In press). "Concept Management for Terminology: A Knowledge Engineering Approach". *Proceedings of the ASTM Symposium on Standardizing Terminology for Better Communication: Practice, Applied Theory, and Results*. Cleveland, Ohio, June 1991.

Meyer, Ingrid. (1991). "Knowledge Management for Terminology-Intensive Applications: Needs and Tools". *Proceedings of the Workshop on Lexical Semantics and the Lexicon* (Berkeley, June 1991), Ed. James Pustejovsky and Sabine Bergler. Association for Computational Linguistics

Meyer, Ingrid, Onyshkevych, Boyan and Carlson, Lynn. (1990). "Lexicographic Principles and Design for Knowledge-Based Machine Translation". *Technical Report CMU-CMT-90-118*, Centre for Machine Translation, Carnegie Mellon University, Pittsburgh.

Meyer, Ingrid and Paradis, Line. (1991). "Applying Knowledge-Engineering Technology to Terminology: A Pilot Project". *Terminology Update*. Ottawa (Canada), Dept. of the Secretary of State of Canada, pp. 3-8.

Meyer, Ingrid and Skuce, Douglas. (1992, in press). "Computer-Assisted Concept Analysis for Terminology: A Framework for Technological and Methodological Research". *Proceedings of the Fourth International Congress of the European Association for Lexicography (EURALEX)*, Malaga, Spain, 1990.

Moller, J-U. (1988). "Knowledge Acquisition from Texts". In: Boose, J., and Gaines, B. (eds) *Proceedings of the European Knowledge Acquisition Workshop*, pp. 25-1 to 25-16.

Motta, E., Rajan, T. & Eisenstadt, M. (1990). "Knowledge acquisition as a process of model refinement." *Knowledge Acquisition*, 2(1), 21-50.

Moulin, Bernard and Rousseau, Daniel. (1990). "A Knowledge Acquisition System for Analysing Prescriptive Texts". *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.

Murray, Kenneth and Porter, Bruce. (1989). "Controlling Search for the Consequences of New Information During Knowledge Integration". *Proceedings of the 6th International Workshop on Machine Learning*.

Parent, Richard. (1989). "Recherche d'une synergie entre développement linguistique informatisé et systèmes experts: importance de la terminologie". *META*, 34(3).

Pogson, Geoff. (1988). "Controlled English: Enlightenment Through Constraint". *Language Technology*, 6, pp. 22-25

Reimer, Ulrich. (1990). "Automatic Knowledge Acquisition from Texts: Learning Terminological Knowledge via Text Understanding and Inductive Generalization". *Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, pp. 27-1 to 27-16.

Rondeau, Guy. (1984) Introduction à la terminologie (2nd Ed.) Chicoutimi (Canada): G. Morin.

Sager, Juan. (1990). A Practical Course in Terminology Processing. Amsterdam/Philadelphia: John Benjamins.

Shadbolt, N.& Wielinga, B. (1990). Knowledge based knowledge acquisition: the next generation of support tools. In: *Current Trends in Knowledge Acquisition.* B. Wielinga, J. Boose, B. Gaines, G. Schreiber, M. Van Someren (eds.), Washington, DC.: IOS Press.

Shaw, Mildred and Gaines, Brian. (1988). "KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices". In: *Knowledge Acquisition Tools for Expert Systems*, J. Boose and B. Gaines (eds), Academic Press.

Shaw, M.& Gaines, B. (1989). Comparing conceptual structures: consensus, conflict, correspondence and contrast. *Knowledge Acquisition*, 1, pp. 314-363.

Skuce, Douglas. (1991a). A Wide Spectrum Knowledge Management System. Submitted to: *Knowledge Acquisition.*

Skuce, Douglas. (1991b). *Cleartalk Design Notes.* AI Laboratory, University of Ottawa, Canada.

Skuce, Douglas. (1989). "A Generic Knowledge Acquisition Environment Integrating Natural Language and Logic". *Proceedings IJCAI Workshop on Knowledge Acquisition* , Detroit, August 1989.

Skuce, Douglas and Meyer, Ingrid. (1990a). "Concept Analysis and Terminology: A Knowledge-Based Approach to Documentation". *Proceedings of the Thirteenth International Conference on Computational Linguistics* (COLING 90), (In Czap and Nedobity, eds, 1990), pp. 56-58.

Skuce, Douglas and Meyer, Ingrid. (1990b). "Computer-Assisted Concept Analysis: An Essential Component of a Terminologist's Workstation". *Proceedings of the Second International Congress on Terminology and Knowledge Engineering Applications (TKE 90)*, Frankfurt: INDEKS Verlag, pp. 187-199.

Skuce, Douglas and Monarch, Ira. (1990). "Ontological Issues in Knowledge Base Design: Some Problems and Suggestions". *Proceedings of the 5th Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Banff, Oct. 1990. Also available as *Technical Report CMU-CMT-90-119*, Centre for Machine Translation, Carnegie Mellon University, Pittsburgh.

Skuce, D., Wang, S. & Beauvillé, Y. (1989). "A generic knowledge acquisition environment for conceptual and ontological analysis." *Proceedings of the 4th Knowledge Acquisition Workshop*, Banff, Alberta.

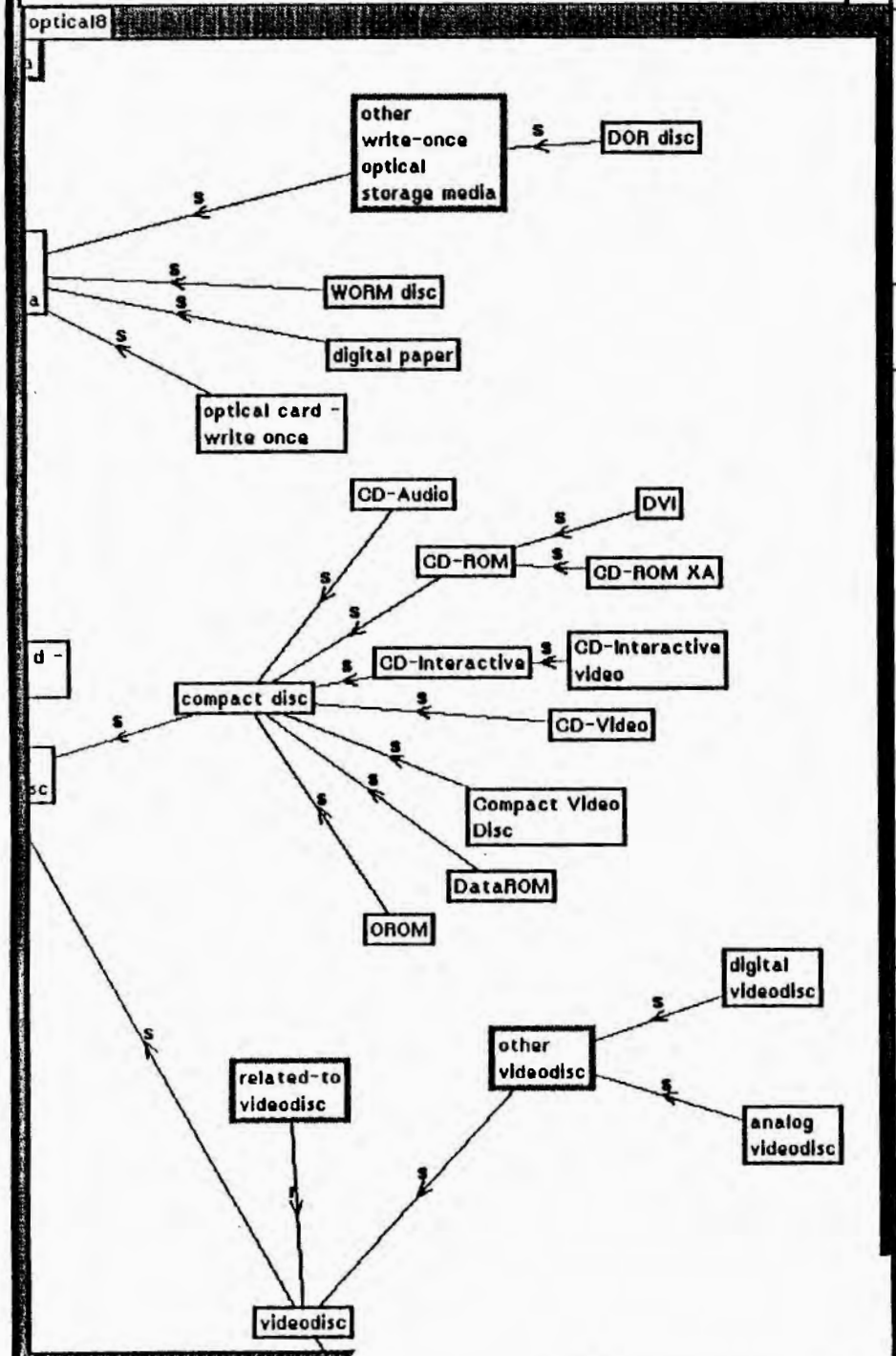
Slator, Bruce. (1989). "Extracting Lexical Knowledge from Dictionary Text." *Knowledge Acquisition* 1, pp. 89-112.

Szpakowicz, S. (1990). Semi-automatic acquisition of conceptual structure from technical texts. *International Journal of Man-Machine Studies* 33, pp. 385-397.

Wijnands, Paul. (1989). "Systèmes experts et terminologie". *META, 34, 3.*

Woodward, B. (1990) Knowledge acquisition at the front end: defining the domain. *Knowledge Acquisition* 2, pp. 73-94.

CODE SYSTEM V.2.4, Copyright (c) 1991 University of Ottawa, all rights reserved



CD-ROM in "pp" format

cdName: CD-ROM
super: compact disc
hasPropsOf:
kinds:
subConcepts: CD-ROM XA, DVI
inheritPropsTo:
instanceOf:
instances:

SPECIAL
classification status: classified
creation date: 3 July 1991
done by: eck
graphic support:
KBName: optical storage
last update date: 22 August 1991
level:
status: nil

DEFINITION
intensional definition: A CD-ROM is a compact disc that can be written once, and read many times, and that may contain any combination of text, graphics, audio and video.

CONCEPTUAL INFORMATION
content: data and digital audio, graphics and still pictures
degree of writability: read-only
dimensions/diameter: 4.72 inches (12 cm)
encoding method: digital
error correction: yes: CIRC (Cross-Interleaved Reed-Solomon Code) or LECC (Layered ECC (extra error correction coding that is layered over CIRC)
Layered ECC, followed by CIRC
introduction data: 1983
material:
observation: CD-ROM has to be used as an adjunct to mini- and personal computer memory systems, i.e. it is used as a computer peripheral; thus, the key to the use of the enormous amount of data storage on CD-ROM is the retrieval software
physical form: disc
recording technology: optical
rotation technique: CLV
standard: Yellow Book (physical format) and High Sierra Group Standard/ISO 9660 (logical format)
storage capacity: user data: 543 MB , total: 600 MB, user data: 550 MB

LINGUISTIC INFORMATION
English collocation/coll1: to pre-master CD-ROM discs
English synonym/syn1: Compact Disc Read-Only Memory
English synonym/syn2: CD-ROM disc
English synonym/syn3: CD ROM disc
English synonym/syn4: Compact Disc Read Only Memory
English term: CD-ROM
English textual support/cont1: "Compact Disc Read Only Memory is a set of standards for storing 600 Megabytes and above of digital information on a Compact Disc. It was developed by Philips and Sony, and is defined in the Yellow Book."
English textual support/cont2: "The Compact Disc Read-Only Memory is based on CD-audio technology. It is a version of compact disc standard used primarily to store general-purpose digital data for personal computers.
French synonym/syn1: disque compact ROM
French term: CD-ROM

Figure 1. A COGNITERM entry

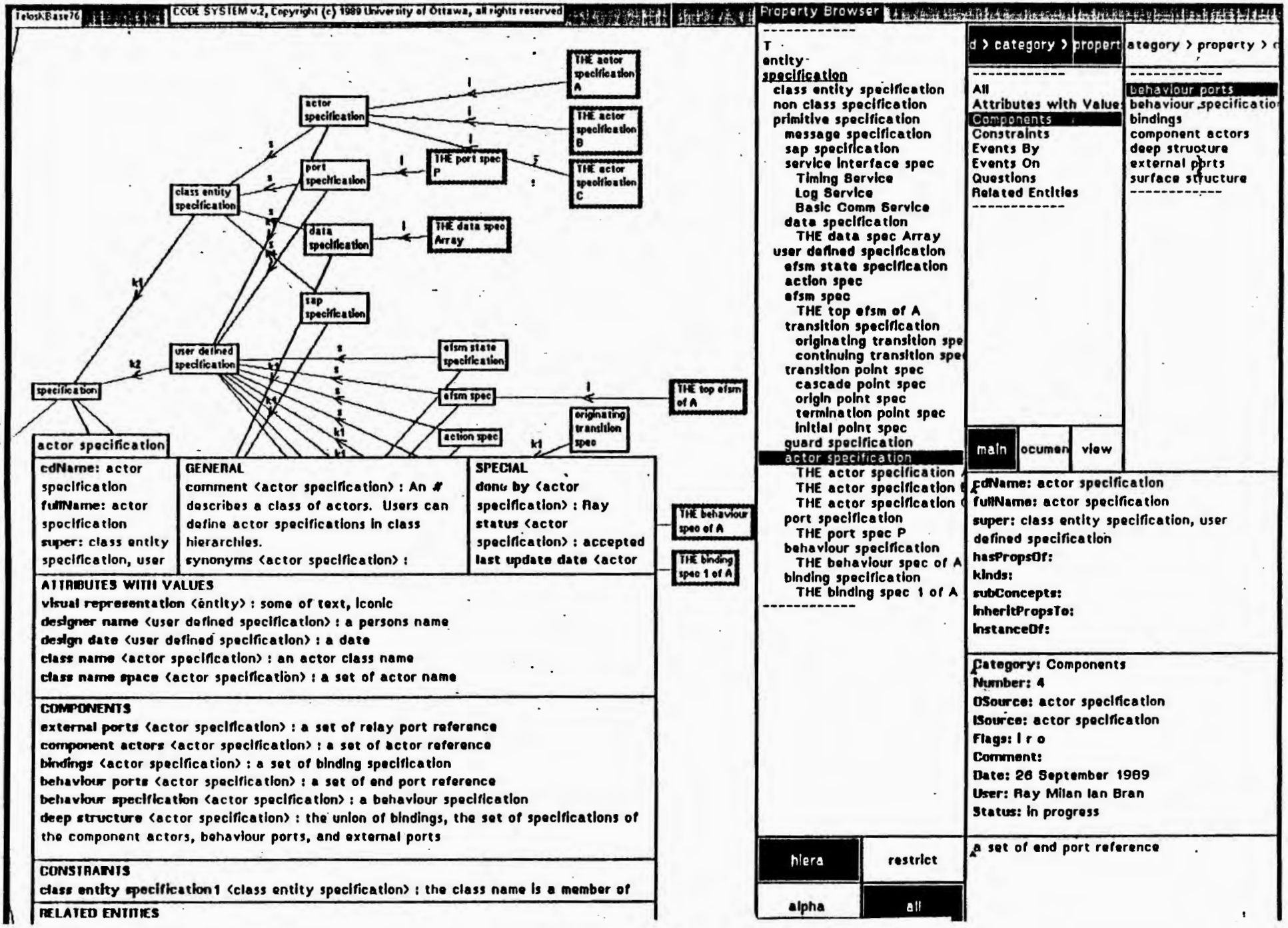
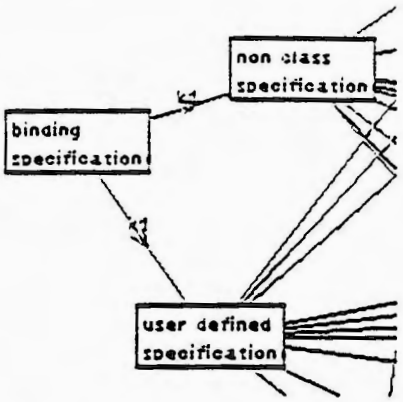


Figure 2 - Full CODE screen of part

Binding specification



A binding specification is a non class specification and a user defined specification. It is an abstraction for a connection between two actors, achieved through ports, and supported by the communication service. An instance of a binding specification is THE binding spec of A.

A binding specification can have the following ATTRIBUTES WITH VALUES:

— a visual representation, which is iconic

— a designer name, which is a person's name

— a design date, which is a date

A binding specification can have the following RELATED ENTITIES:

— it is a component of an actor specification that a binding specification is part of

— an incarnation of a binding specification, which is a subclass of binding incarnation

— ports, which are a set of two port references.

Figure 3. Pseudo natural language output of the cd *binding specification.*

Knowledge Acquisition for Case-based Consulting

Brian M. Slator
Kerim C. Fidel

Institute for the Learning Sciences
Northwestern University
Evanston, IL 60201

Abstract

TaxOps, the Tax Opportunity Advisor, is a case-based system, with both training and on-the-job applications, developed for use in the tax accounting/consulting domain. TaxOps is an example of the "Ask System" teaching architecture. This paper describes the process of knowledge acquisition and knowledge engineering for TaxOps, including interviewing experts, content assessment, indexing, and knowledge representation design, along with the lessons and principles learned during the course of the project.

1. Introduction

The stars of the corporate business world are highly trained, highly paid experts who solve the hardest problems, land the biggest clients, make the biggest deals, and generate the largest returns on their time. These executives are typically career oriented, highly motivated, and very difficult to get hold of on the phone. These executives, as individuals, are important to the success of any corporation, and as a group they form the core of what is sometimes called the corporate memory: a repository of corporate wisdom.

Every large corporation has their share of stars. One problem is that talented people are also mobile, and portions of the precious corporate memory are likely to be lost at any time. Another problem is that corporate stars are often so busy and goal-directed that they do not take time to associate with each other. Important opportunities are lost, particularly in large corporations, because the major players might not even know one another. In other words, corporations soon grow so large that they can not execute the "shout down the hall" collective problem solving strategy.

TaxOps, the Tax Opportunity Advisor (Schank, 1991), is designed to address these issues. TaxOps is a case-based consulting system that engages in a user-directed interactive video dialog on the subject of taxes as business opportunities. TaxOps presents short video clips to the user at a point in the dialog when the user is interested in, and prepared for, the content of each clip. This is accomplished by constructing a network of video clips with links formed by matching questions raised by each clip with questions answered by other clips. Knowledge acquisition in this domain is accomplished in two stages: 1) a set of tax experts are interviewed on videotape to elicit domain knowledge in the form of stories, advice, and anecdotes; 2) the video clips are summarized and annotated with questions raised and answered. The collective knowledge of a group of highly successful tax experts is then available to the user, both as explicitly recorded on videotape (and later pressed onto laser disk to allow high-speed random access), and as implicitly encoded in the questions that link the clips together into a network.

Part of the theory behind TaxOps is that experts represent their expert knowledge in terms of "war stories" that embody their most important experiences from which they reason, and that experts who are good teachers tell their war stories in a way that is more meaningful, more instructive, than simply enumerating a set of principles (Schank 1982). TaxOps is a tool for communicating these stories.

To initiate a TaxOps session the user fills out a short form, describing a business situation of particular interest. This form is either used to describe a specific business, such as the Duet Cup Company, a paper products manufacturer with net revenues of $382 million and clear credit history, or it can be used to describe a generic business case, such as a pharmaceutical manufacturer with cash flow problems operating in several U.S. states. Once the situation has been described, and subsequently analyzed by TaxOps, the user is given a choice of several different entry points to the video clip network. These entry points correspond to clips whose subject matter is most directly relevant to the business case. After entering the system in this way, users are free to navigate through the video network, viewing clips that are relevant to their purposes, and traversing to other clips along links that pose questions that are answered by connected clips. The session is guided by the structure of the network but controlled by users whose interest in the subject matter leads them from clip to clip.

TaxOps currently runs on a MacIntosh; the system is written in Common Lisp. A total of twelve experts were interviewed, seven of the interviews were committed to laser disk. The system contains approximately 500 individual video clips, totalling nearly four hours.

2. Deliverables -- Advisory vs Expert Systems

Expert systems make a precise diagnosis and suggest specific remedies. Advisory systems like TaxOps are intended to provide fairly general advice, and to provide background, alternatives, and context. The advice is intended to prompt users to exercise their creativity by giving examples of the kinds of things that can happen and the kinds of services that can be provided.

It is hard to gauge TaxOps's "results". When some systems might give advice of the form *"When there is a threat of hostile takeover institute a personal financial planning (PFP) program for top executives"*, TaxOps, instead, offers a clip in which an expert tells the story of an actual hostile takeover and how it was dealt with. This would, very likely, include something about the necessity of providing PFP service, but that advice would be given in the context of a real scenario, and would be presented less as an if/then rule, and more as an explanation: *"When there's a threat of a hostile takeover executives often feel nervous, and they'll want to know that their personal finances are in order"*. The user can infer the rule (provide PFP when there is a threat of hostile takeover), and the expectation is that the "rule" is better internalized because of its practical applications, and because the rule operates in the real world, as demonstrated by the "war story".

TaxOps's advice cannot be "wrong" (provided that the experts tell their stories accurately); it cannot mis-diagnose since it does not provide a definitive opinion, but rather identifies relevant avenues of enquiry. TaxOps would, however, fail if its users did not understand the relevance to their current problem to the clips they were offered. TaxOps would also fail if it could not offer users the right clip at the right time, including diverging paths for the user to follow at will.

3. Directed Interviewing

The first stage of knowledge acquisition for TaxOps was accomplished by interviewing a dozen senior partners at Arthur Andersen & Co. (AA&Co.), one of the "big six" accounting firms. The partners were selected from various tax practices: State and Local Taxes (SALT), Real Estate Consultation, Appraisal and Valuation, Personal Financial Planning, and others. The interviews were conducted over a period of several weeks. Each interview generally lasted from two to four hours and yielded between fifty and one hundred usable clips (where the expert was able to relate a story or express a complete thought without having to pause or start over). These clips ran from thirty seconds to four minutes in length.

The interviews were conducted in an exploratory fashion. It was not paramount that the video clips from any interview be cohesive or externally thematic, rather it was important that as

many good stories as possible be collected. The interviews were, therefore, somewhat informal and conversational. The purpose of the interviews was to collect a set of business "war stories" that could be shared with other senior partners as a way of encouraging communication and cooperation among members of the various service lines. The TaxOps video library was also constructed to support a tool for computer assisted training for junior members of the firm. As the interviewing progressed, certain recurring themes were identified, and these were pursued in future interviews. For example, in light of the recent world economic picture, many of the partners had stories relating to "downsizing" or other strategies for corporate retrenchment and cost cutting. Since each service line provides services complementary to those of the other lines, each partner was able to cast light on business situations from the perspective of their specialty. The Appraisal expert, the Real Estate expert, and the Personal Financial Planning expert could, for instance, suggest ways to think about hostile takeovers that might not normally have occurred to experts in State and Local Tax.

Another theme that emerged in the interviews was the notion of the "key indicator". Each partner was asked to talk about key features of business situations that indicated an opportunity for their group to provide service. These ranged from the predictable (i.e. any time there are transactions where the dollar values are large, experts in appraisal and valuation should be consulted), to the obvious (i.e. a company planning to relocate to another state needs the services of the State and Local Tax group), to the unexpected (i.e. clients that operate in only one state risk significant tax exposure if they do not consult a State and Local Tax expert - this is because SALT's practice is thought of as primarily having to do with multi-state clients). Similarly, each partner was able to reel off "key questions" to ask prospective clients in order to determine whether opportunities existed to sell services from that partner's service line.

Although the interviewer's voice was audible on the tape, no effort was made to incorporate the interviewer's questions into the links (questions raised to questions answered) that eventually connected the clips in the video network. The links were formed by having tax professionals view the clips, with the interviewer's voice edited out, and generate the questions they thought each clip answered and raised. This task was executed by senior managers, one step below partners in the AA&Co. hierarchy.

The tax experts were interviewed by a non-expert, and this influenced the way that answers were framed. The rules of conversation (Grice 1975), require an expert to voluntarily define terms and flesh out background issues while in conversation with a non-expert. This is a matter of courtesy to the non-expert, although it usually goes unnoticed. When an expert views footage of an expert speaking to a non-expert it is clear to them (and sometimes annoying), that seemingly "obvious" things are being spelled out for the benefit of a non-expert. This did not effect the pragmatic value of the clips' contents but it did effect the delivery and thus the user's perception of the clips to some extent.

4. The Need for an Index

TaxOps was conceived of as a system where users would watch a clip that would presumably raise questions in the their minds and these would be anticipated. Similarly, each clip would also answer questions and these would be enumerated. A network would then be constructed by searching for matching questions, weaving the library of clips into a hardwired network. It soon became clear, however, that the subject matter made it difficult for non-experts to link questions with confidence. Perhaps the most important problem was that the number of questions answered and raised in the space of one interview's worth of clips made it intractable to search the entire list of questions answered to find the best matches for each question raised. Without the addition of some automated process to help choose links it would be impossible to handle the task of linking networks in a timely or efficient way.

An index is a scheme for categorizing the elements of a domain. Because of the scale of the TaxOps domain, an index was derived by looking at the set of questions answered and raised

that had been collected and finding what categories the questions seemed naturally to fall into. In addition, a few aspects of the index were "artificially" accentuated. For instance, hostile takeover was a subject that came up a number of times, so provision was made in the index to highlight anything that had to do with hostile takeovers.

The TaxOps index is neither deeply semantic nor deeply feature-based. It merely attempted to describe at a high level the subject of each question posed (i.e. "cash-flow problem", "you should bring us in when..."). The index did not attempt to describe clips in great detail. In part this was because it is impossible to capture every possibility. In addition, a large, complex index would be undesirably cumbersome. The emphasis is placed on the questions when choosing links, using the indices only as preliminary clues to the content of the clip. The index was designed in a way that allowed a question to be "multi-indexed", that is, to be identified as simultaneously a member of several distinct branches of the index tree. This helped to build a picture of the questions without having to develop extremely leafy branches. On the other hand, domain constraints informed the structure of the index -- the index reflects only the business opportunity domain.

5. Application of the TaxOps indexing system

The following describes the process of indexing for TaxOps, beginning with unedited videotaped interviews, and discusses some lessons learned during the development of the TaxOps indexing process.

5.1 Choosing usable clips

The first step in indexing (defined broadly as the process of classifying the information in the program's video library both for retrieval and summary representation to the user) is to view videotaped interviews with experts in order to break the continuous interviews into sets of separate clips.

The task of identifying clips requires some judgement on the indexer's part. Sometimes it is necessary to allow repetitious material in order to preserve the overall delivery of the clip. Experts sometimes talk about several subjects in a story. Sometimes it is possible to separate them, sometimes it is necessary to let them run together. It is, in general, most helpful in later stages if each clip contains information on one subject or a story describing one experience, and clips more than a minute long tend to discourage users, though many good stories do run for two minutes or more.

5.2 Content Analysis

Once usable clips are identified, edited versions of the interview are produced containing only the clips, each clip separated by three seconds of blank tape. These clips are indexed (that is, they undergo content analysis).

5.2.1 Personnel

Content analysis for TaxOps' video library was originally performed by institute personnel who were neither the intended end users of the system nor experts in the domains the stories dealt with. This was unsatisfactory. A principle of indexing seems to be that highly specified domain knowledge intended for use by specialists in the same or similar domains cannot be adequately represented by novices in that domain. In other words, to classify and transmit expert knowledge and wisdom about a specialization of AA&Co's tax practice to an expert in a closely related field it is necessary to have content analysis done by someone sufficiently knowledgeable in the field, someone representative of the end user group. Ideally, the person performing content analysis on a clip by an expert in Real Estate Consultation, for example, would be an expert in some field (within the larger domain of AA&Co's practice) other than Real Estate Consultation. It is unlikely that experts in Real Estate Consultation will need to

view clips dealing with their area of expertise, and experts analyzing clips in their area of expertise might take some things for granted (particularly in posing questions raised) that an expert in a different field would need to have explained in more detail.

There is some leeway in this. In essence it is important that the indexer (content analyzer) be sufficiently knowledgeable to be able to recognize the content of clips at the same level as end users and anticipate the questions an end user might have, keeping in mind one's bias from specialization. This task was carried out well by a pair of senior managers who were good "generalists" in their knowledge of the range of AA&Co services and, of course, experts in their own field of specialization.

5.2.2 Synopses

It is unlikely that an indexer will be able to remember and process all the details of a complex clip after watching it once or twice. To continuously go back and view the video takes a long time. The number of times a video needs to be reviewed has, in our experience, been greatly reduced when indexers write out synopses as the first step in content analysis. Reproducing stories may aid memory and saves the time of watching the story over and over.

Synopses, which are narrative, are particularly important for narrative stories. They are of little importance for merely expository clips (the points made in the clip then become focal - see below). The synopsis retells a clip's story in skeletal form (usually one, two, or at most three sentences), giving a good idea of the content of the clip. When a user indicates interest in a narrative clip (usually because they have been shown a question the clip answers) TaxOps shows the user a synopsis of the clip. Based on seeing the synopsis the user will decide whether the clip is sufficiently interesting to watch. It is conventional to predicate synopses of narrative stories with the phrase "A story about". The following fictional examples illustrate the application of synopses:

Example 1: Narrative Clip

> *"We had a client who was having trouble dealing with problem Y. Many clients think they should deal with Y problems themselves, but you really need the expertise of Group X to deal with them properly. We had been performing some small services for this client, nothing major, but one day I happened to hear about this Y problem they had. I said to the Vice-President "Let us come in to have a look at your books; maybe we can spot the source of the problem." He agreed. Sure enough, as is often the case with Y problems, they were making the typical mistake, Z. We were able to help them correct this problem, and have been performing regular checks for Z ever since. The client has been so pleased with our service that this has led to additional and ongoing work for groups A, B, and C as well. You should know if your client might have a Z problem. If they do, or if you think they might soon, you definitely need to call Group X."*

A synopsis of this story might be the following:

> A story about a client who was experiencing a Y problem. Group X was able to find and help correct the problem and has provided ongoing services since then, which have in turn led to opportunities for other groups.

Example 2: Expository Clip

> *"The personnel at group X are all experts in the area of Y problems. We can handle almost any Y-related issue. Group X can get the necessary resources to your client in any part of the country, and we believe we can add significant value for your client, no matter how large or small the Y problem. None of our competitors can bring such extensive resources to bear on a Y problem. So if you think your client might have a Y*

problem give us a call. We'll help you evaluate the opportunities and determine whether Group X can add value for your client."

A synopsis for this kind of clip will be much less expressive, as in the following:

> A discussion of Group X's resources for dealing with Y problems.

From the examples above, which are fairly representative of the kind of content the TaxOps clips presently contain, it is clear that synopses are more appropriate for capturing the sense of narrative stories but are not very helpful for describing the contents of expository clips.

5.2.3 Points

Points represent the content of clips in expository form. When a clip is expository, users are shown the points made in a clip to help them determine whether they want to view the clip.

A specificity problem similar to that encountered in posing questions (discussed below) occurs in formulating points. Points need to be informative and the sum of points for an expository clip need to represent the content of the clip well, but it is unnecessary, and probably impossible, to reproduce the entire contents of the clip. Points should capture the gist of the clip. Given that clips are one or two minutes long, between one and four points, of one and occasionally two sentences in length, capture the essence of a clip.

The rule of thumb for point writing is to sum up in a couple of sentences what the expert is trying to communicate in the clip. Points can be thought of as being like notes taken at a lecture. Points for the fictitious example clips in the section above might be as follows:

Example 1 (Narrative clip): Points:

1. Clients often try to deal with Y problems themselves.
2. Group X must be called when there is a risk of Y problems.
3. Y problems are often caused by Z.
4. Service of Y problems by Group X can lead to opportunities for other groups.

Example 2 (Expository clip): Points:

1. Group X has resources and expertise to deal with any Y problems throughout the country.
2. Y problems of any size are good opportunities for group X to add value.
3. Group X's expertise and resources for dealing with Y problems surpasses those of any competitor.
4. Group X will help the engagement team determine need for Y-related services.

5.2.4 Posing Questions

In keeping with the tradition of "Ask" systems (Schank 1991; Schank et al., 1991), TaxOps indexing is applied to questions raised and answered for each clip. The questions that are presently used in TaxOps were written with an index in mind, and the AA&Co. managers who wrote the questions were required to index their questions as they posed them. We now believe that this was a mistake. If the person posing the questions has an index in mind, before long the index will determine the questions posed. This has several detrimental effects. The question poser will be discouraged from posing questions that do not fit within the index, and the questions he does pose will be little more than restatements of the index category that applies to the question. Our experience is similar to that of the Protos project:

> "Formulation of an explanation for Protos is, of necessity, a two-step process. The teacher must conceive the explanation in his own terms and then must translate it into Protos' language. Performance of this knowledge engineering task proved to be difficult for the teacher. *He tended to skip the first step and to try to formulate*

explanations directly in the explanation language." (Bareiss 1989, pg. 80, emphasis added)

Though TaxOps does not require the indexer to translate the precise meaning of a question into a language comprehensible to the program, we found questions determined by the index to be particularly problematic for questions raised, because these questions refer to content outside the clip. Questions answered draw their content from the expert's stories and so are less subject to external determination (though the problem persists here as well).

Those who pose questions should be allowed to pose them without immediate reference to an index. They may be encouraged to pursue certain lines or types of questioning (see below) but the questions should not be determined by some external structure. Rather, since the goal is to record accurately the questions a clip answers, and to anticipate the questions a user would want answered next after seeing a clip, spontaneity on the part of the indexer/question-poser should be promoted.

5.2.5 Questions Answered

Each clip's contents are described in terms of the questions the clip answers. In other words, every useful piece of information or wisdom given by the featured expert can be seen as the answer to a question a user might ask. Each clip answers at least one question and can answer many questions. We found three questions to be the typical number needed to describe an average clip.

Question posing can be problematic. It is hard to know at exactly what level of specificity a question should be posed. If a clip contains the statement "group X provides Y service" one could pose the question this statement answers in several ways. A non-specific question answered might be *"What is a service that group X provides?"* while a more specific formulation could read *"Does group X provide service Y?"*.

Both formulations have their drawbacks. If the non-specific form of the question is employed and group X provides many services, the clips that describe these services will all have the same question answered and there will be no way to distinguish between clips when choosing what to offer the user next (or when the user who sees the questions is selecting what to see next). On the other hand, posing the specific question risks making a mockery of the question asking method. If every time an expert says *"We provide service Y"* we attach *"Does group X provide service Y?"* the user will quickly catch on to the fact that this "question" is in fact a statement that service Y is indeed provided. Posing questions in "Jeopardy" form is likely to discourage rather than motivate the user.

When a question of this nature arises it is best to aim for fairly high specificity but to avoid the "Jeopardy" tenor if possible. If all the expert says is *"We provide service Y"* one is probably stuck with *"Does group X provide service Y?"*, but most often such a statement is accompanied by additional information, either a story or expository remarks, and questions like *"How does group X provide service Y?"* become available. Such a question does indicate service Y is provided but also lets the user know that there is more to the information offered about service Y than just the fact of the service's availability. Occasionally a clip will contain a list of services (or some similar general statement) with little or no exposition, in which case a general question like *"What services does group X offer?"* may be appropriate.

The indexer must exercise judgement with regard to posing questions answered. If a clip contains just a passing mention of some subject and deals with other topics in detail it is most often appropriate to pose questions only regarding the main topics of the clip. If the user chooses to view a clip based on having been shown a question that clip purports to answer, the user needs to be satisfied that the clip does indeed answer that question in a meaningful way.

This generally means questions answered should be posed regarding topics given prominence in the clip.

There should be at least one point in each clip that addresses (not necessarily completely) each question answered for that clip, and every point enumerated for a clip should have at least one question answered that points to it. A question answered can apply to more than one point and a point can apply to more than one question answered.

Returning to the example clips from above, the questions answered might be the following:

Example 1 (Narrative clip): Questions Answered:

1. Should clients try to deal with Y problems without expert assistance?
2. How does Group X help clients overcome Y problems?
3. What often causes Y problems?
4. What opportunities to provide additional service can result from a successful Group X engagement?

Example 2 (Expository clip): Questions Answered:

1. In what parts of the country is Group X prepared to deal with Y problems?
2. How big does a Y problem have to be to make bringing in Group X worthwhile?
3. How does Group X compare with the competition in dealing with Y problems?
4. How can the engagement team determine whether Group X's Y-related services are needed?

Some of the questions, like question two in the first example, are not directly answered by the corresponding points, but the points do indicate that some additional relevant information is contained in the clip. Other questions, like question two in the second example, are answered by the corresponding points. In general, points correspond to, and can be a source of, questions answered by a clip

5.2.6 Questions Raised

After watching a clip, it is assumed, some questions will arise in the viewer's mind. When an indexer poses questions raised it is an attempt to anticipate these questions. Eventually questions raised for a clip will be matched with similar questions answered from other clips and these links will form the TaxOps video network.

There are problems of specificity that arise when posing questions raised. If a question is highly specific it may be that no adequate answer can be found elsewhere in the network. The network cannot depend solely on such specific questions, though these questions may still be legitimate and can be posed. Sometimes an answer will be available (and the indexer can "cheat" a little by knowing, from watching other clips, that the answer is available). If the highly specific question is very relevant a lack of answers may indicate the need for additions to the video library; this is a useful property of the question-posing methodology: it reveals what knowledge is missing from the network. However, for practical purposes, the network cannot depend on these additions being made.

Highly general questions also can be problematic since there are likely to be too many clips available to answer the question. The rule of thumb in posing questions raised is for the indexer to imagine talking to the expert in the video clip, and that the expert has just given the contents of the clip in response to some question. The question poser should try to imagine what would naturally follow in a real conversation. It is usually wise to include both general and specific questions raised for each clip.

Questions raised should usually cover a range of topics. If the clip dealt mainly with cash flow it is reasonable to pose a question that asks for more information about cash flow. It is necessary from a practical standpoint, however, to pose a question or two about something

else. Questions raised are paths out of a clip. If there is only one path, or one type of path, out of a clip it may constrain users unduly, and if that path out is identical to the path in (as would be most likely in the example above) the user would never have a chance to get off the subject of the clip. When a clip deals with one or two well-defined topics (such as how group X deals with cash flow problems) it is reasonable not only to ask for more information about cash flow problems, but to also ask what else group X can help with. Such a combination (with both fairly specific and general aspects) allows the user to continue a line of inquiry or to move on to something else.

Questions raised need to be posed in a somewhat disciplined fashion. One could potentially raise questions about any clip ad infinitum. Once again, the rule of thumb is to pose those questions a user (for TaxOps, a highly knowledgeable business professional investigating a real business opportunity) might ask as a follow-up to the information a clip imparts. We found that three or four questions raised were appropriate to most clips.

Occasionally indexers will encounter problems thinking of questions raised for some clips. This can occur because a clip might seem to be so complete in itself that nothing else needs to be said, (and it sometimes occurs due to numbness brought on by indexing too many clips at a stretch). It is important to keep in mind that a user may arrive at any clip via many paths. The clip may be a digression for the user, one of several lines of inquiry, or a component in a larger search. When all else fails there are default questions, some of which are nearly always bound to be appropriate. Examples of these are questions like *"When (else) should you be called?"*, *"What (else) do you do for clients?"*, and *"What should I look for to identify service opportunities?"*. As discussed above, it is desirable to provide at least two different paths out of clip.

Some questions raised that might be applicable to the example clips discussed above might be the following:

Examples of Questions Raised:

1. What are other problems that need expert assistance which clients often try to deal with themselves?
2. What are the signs of a risk of a Y problem?
3. What are other problems Group X specializes in solving?

These questions facilitate links to other clips that deal with the same topic (question 2), different topics with similar themes (question 1), and to completely different topics (question 3). They are all sufficiently general to be likely to have good answers available in the system.

5.2.7 Indexing Questions

Once content analysis (enumerating a synopsis, points, and questions) has been performed on a clip the clip's questions are indexed. The present index (figure 1) has been applicable to all questions TaxOps indexers have posed. If "Y problems" correspond to AA&Co. opportunity indicator 101 (where 6 is the indicator for "hostile takeover" and 50 is for "downsizing", and if Group X offers service line 3, see figure 1), indices for some of the questions given in examples above are:

Questions Raised:

What are the signs of a risk of a Y problem?
Index: 2A2.101
What are other problems Group X specializes in solving?
Index: 1C3, 2A2, 2B

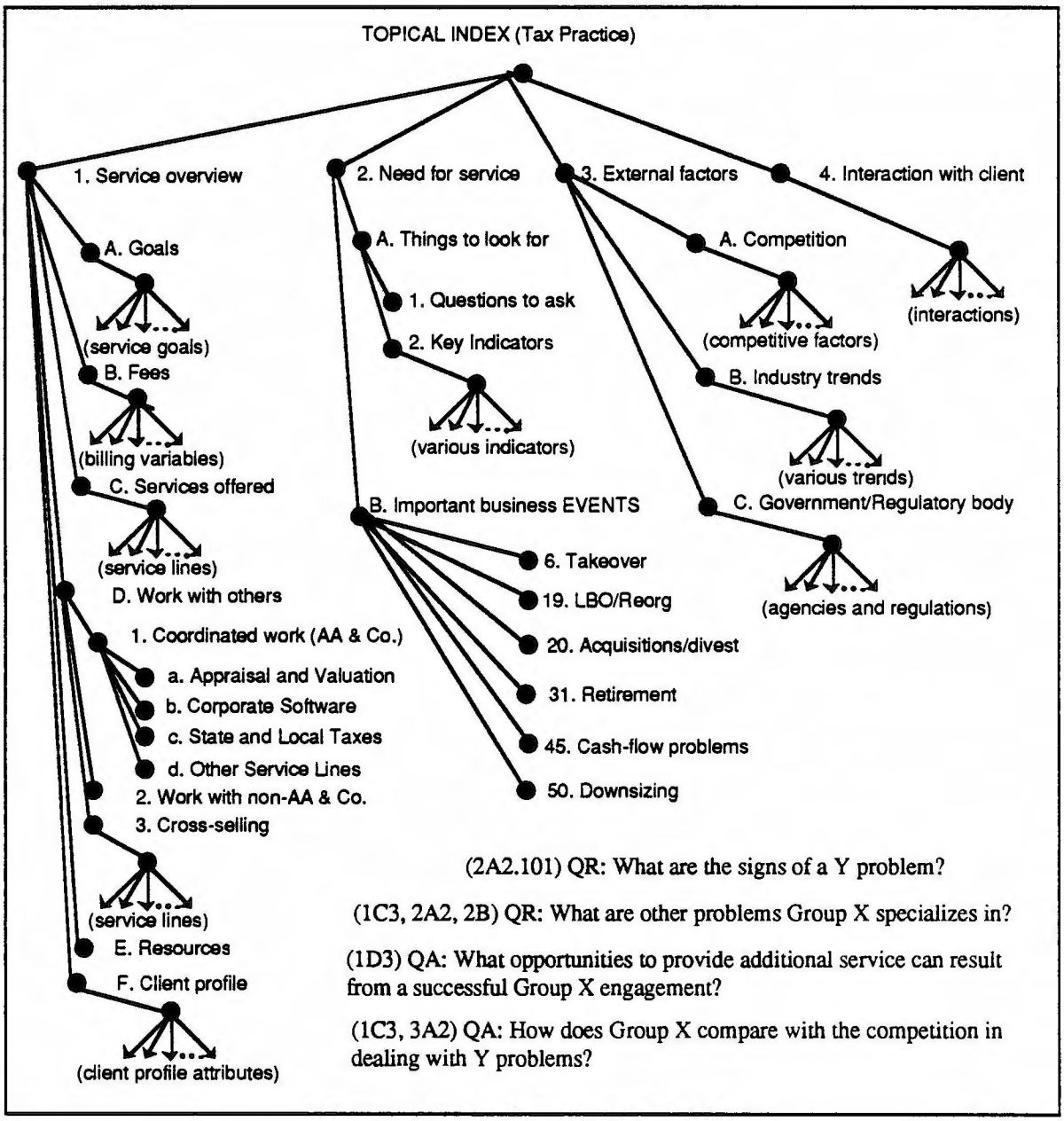


Figure 1: The TaxOps index (a partial representation, showing approx. 10% of the structure, with questions-answered and questions-raised arranged below). The questions are assigned index codes, and are related to each other by their position in the index structure..

Questions Answered:

What opportunities to provide additional service can result from a successful Group X engagement?

Index: 1D3

How does Group X compare with the competition in dealing with Y problems?

Index: 1C3, 3A2

6. Knowledge Representation and Structure

The TaxOps video network is constructed by linking clips together on the basis of matching questions raised in one clip to questions answered in other clips. In terms of content, the meaning structures associated with a clip are reduced to the questions generated by the clip. Rather than attempt an analysis of the transcript of each individual clip, an analysis is made of the questions according to the content categories that the questions fall into.

The task of assigning questions to categories is accomplished in stages, by inductively developing a categorization or indexing scheme that separates the questions into meaningful groups. Developing an ontology of questions is a knowledge representation task like that faced by any knowledge engineer. The questions associated with clips fall into categories that are determined, at the bottom, by the domain of the system.

In the TaxOps domain, an ontology of four high-level categories emerged from the data: services offered, services needed, external factors, and client interaction. The development of this organizing scheme was incremental, with categories being added and replaced as new insights into the domain came forward. The lower levels of the ontology were supplemented by materials volunteered by AA&Co. For example, the lowest levels in the services offered category are the services that AA&Co. offers to clients. Similarly, the services needed category is augmented by a system of "opportunity indicators" developed to match particular client needs with particular specialists within AA&Co. These indicators ranged from industry conditions, like "capital intensive" to industry trends, like "recent history of hostile takeovers".

Because of the constrained nature of TaxOps' domain and application, the index only needs to capture features that are relevant to business opportunities. Primarily feature-based indexing is employed as opposed to using a highly semantic index. It is more important that two clips be linked because they both deal with hostile takeovers then that they be linked by the fact that they both describe instances of, say, agents experiencing expectation failures. TaxOps' highly goal-directed expert users find cross-contextual remindings inappropriate. Perhaps both an expert cook and an expert tax consultant could make the same point about, for instance, the need for thoroughness, but the TaxOps user will better respond to an expert who deals with similar problems to those the user faces. The index, therefore, facilitates classification of questions according to the practical business conditions to which they are relevant.

6.1 Constructing Networks

The process of constructing video networks from clip descriptions takes place in two stages: building and pruning. Building a network is accomplished by matching the index on a question raised by a clip against the set of indices attached to all the questions answered by the other clips. When a match is found, a link is formed between the clip that raised the question and the clips that answer indexically related questions. Very general matches, between questions indexed high in the index structure are given weak links; the more specific the matching, the more strongly associated the clips. This strength of association is used to select which links will appear on the screen at any given time, since there is often not enough room to display them all.

The index is intentionally "high-level". It does not attempt to specify all possible questions. In part this is because an extremely deep index for such a project can never be complete. Having a shallow index also makes the process of indexing questions a tractable one, though there is a trade-off between time and effort spent applying indices, and the time and effort spent reviewing links suggested by shallow indices.

Because questions are indexed at a high level, roughly half of the links from any question raised suggested by the index will be inappropriate. It is therefore necessary to "prune" the links manually. Questions raised are displayed along with the questions that, according to the

index, should answer them. These links are then either approved or rejected by hand. This process takes the place of extremely deep indexing structure and the added complexity of applying such a structure.

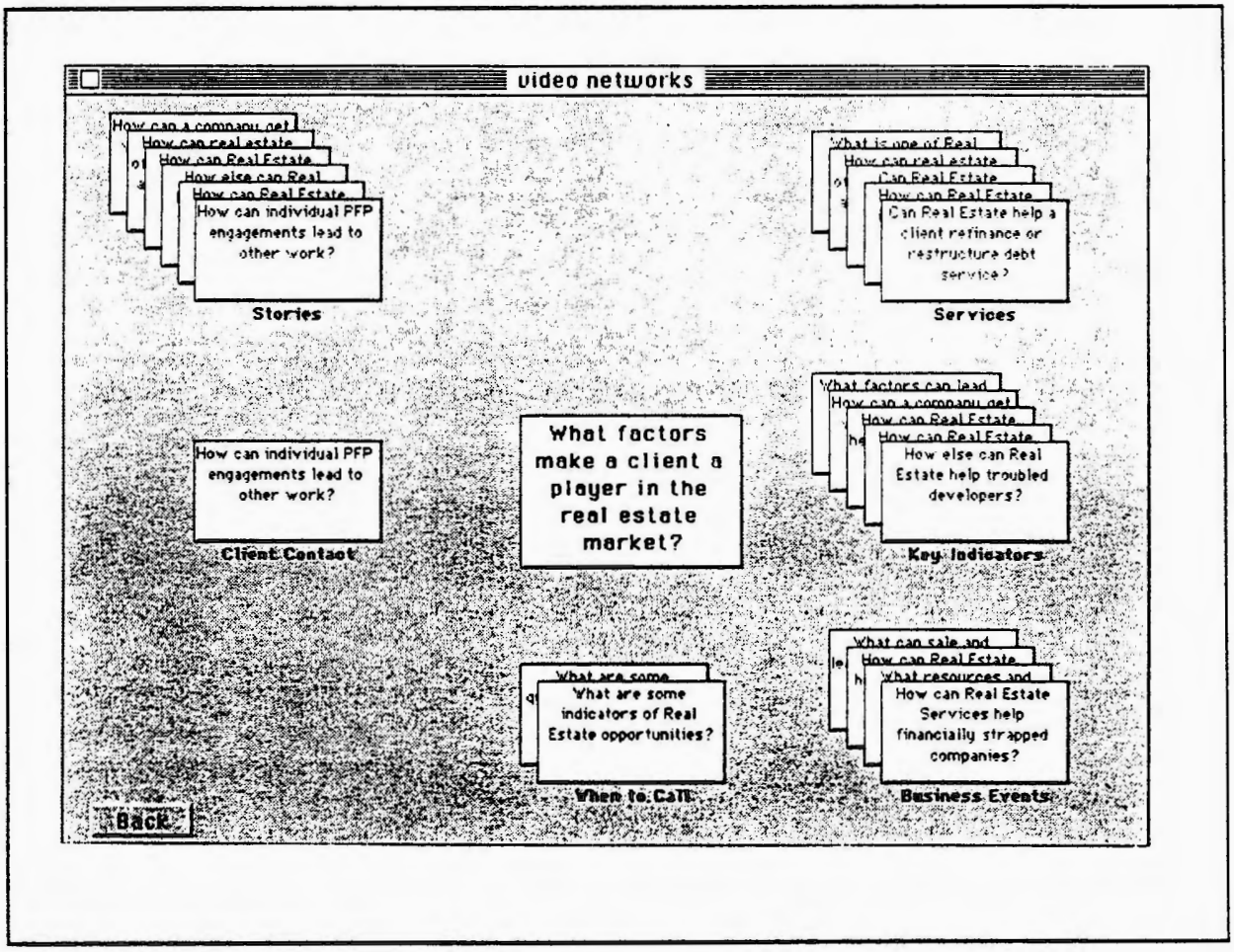


Figure 2: The center card is the focus of the screen. The surrounding cards are questions that may be raised after viewing the video clip that answers the question on the center card.

6.2 Traversing Networks

TaxOps allows the user two views of the video clip network. One view (figure 2) presents a portion of the network as index cards with questions written on them. A clip is associated with at least one card. Users can view the clip if they are interested to hear the answer to the question on the card. The network is traversed by dragging a card from a surrounding pile into the center of the display. This is a browsing interface and it is possible for the user to eventually view any and all clips from any starting point.

The second view (figure 3) presents a more focused perspective on the video network. In this interface the user is presented with a screen that structures the links to other clips according to particular indicators and link types. Stories, for example, are connected to a story button, and follow-up questions are displayed toward the bottom of the screen. In this view, the indicators are ranked according to their relevance to the user's interest, and pressing the buttons marked "1st", "2nd" etc. take the user from screen to screen.

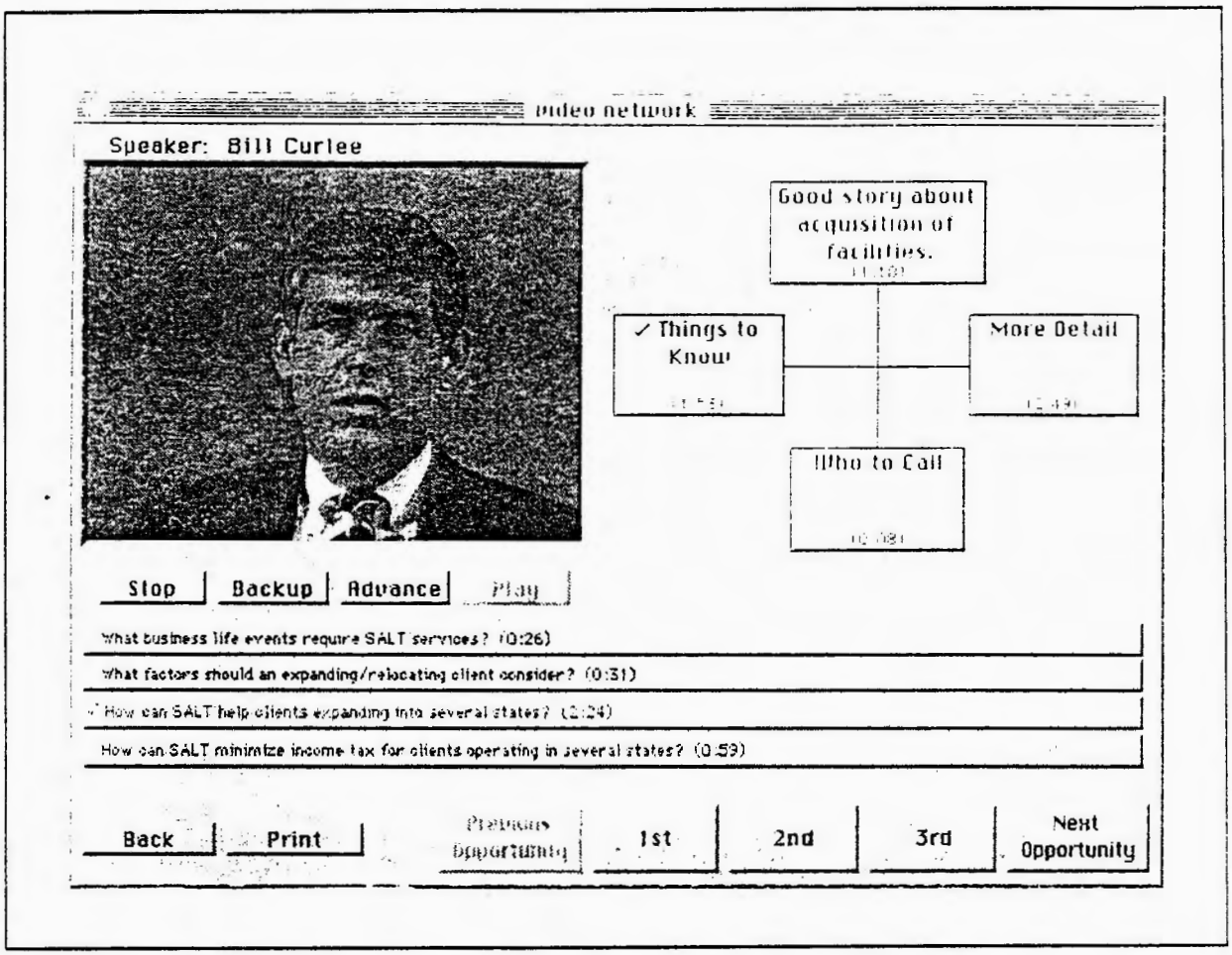


Figure 3 The job aid interface. The upper left corner of the screen is where the video clips are displayed. A summary of the clip is displayed in that region before the user actually views the clip.

7. Results/Future Work

The initial reaction to TaxOps has been favorable. AA&Co. partners and managers are enthusiastic about TaxOps's potential for increasing knowledge and communication across specializations, and the possiblility of building a similar system for other practices is being investigated.

The problem of updating the TaxOps knowledge base has yet to be addressed. Tax laws sometimes change, and trends in Tax Services also come and go. Presently, for instance, hot topics in taxation include EEC '92 (the new European Community trade regulations), mergers and acquisitions, and the depression in the real estate market. In a short period of time these problems may have faded, or changed radically, and TaxOps needs to be able to roll with the changes. The problem is largely one of library management; keeping track of clips that stress time-dependant knowledge and those that deal with consulting "eternals". The problem is also one of getting experts to cooperate. A video taping session is an expensive proposition. Partners bill for their time, both to clients and internally, and will charge whoever sponsors the project for the time they spend interviewing. They also lose time in which they might be servicing clients. It is not realistic to expect partners to grant several interviews. There are additional costs associated with videotaping and production; crews and equipment, not to mention the (relatively nominal) cost of content analysis.

If TaxOps succeeds in increasing AA&Co.'s fees from cross-selling it will quickly more than pay for itself. Because TaxOps's output is hard to quantify and measure, however, it will not be easy to tell if it is actually working (a problem, not coincidentally, faced by education the world over -- costs are easy to assess, but benefits are not). AA&Co. is presently trying to determine whether use of TaxOps can have an appreciable affect on profits. TaxOps may be destined to become a training tool rather than a selling tool. TaxOps fulfills an important function, but its "bottom line" value must be determined.

8. Conclusions

A small set of principles and shortcuts have been revealed as a result of exploring the problem of knowledge acquisition for case-based consulting.

Get away from video as soon as possible. Questions should be posed on the basis of Synopses and Points. Viewing and reviewing clips takes too much time. In addition, having indexers write synopses and points "up front" begins, in an effective way, the process of abstraction, that ultimately results in questions.

Questions should be posed without reference to an index. If the people posing questions have an index in mind when they are posing them, the index completely determines the questions. The concepts of the domain should be allowed to emerge from unconstrained questions. This could eventually create a need to revise the index, but it results in a richer and more useful network.

Writing Synopses, Points, and Questions must be done by a representative of the end-user group. As per the "rules of conversation" (Grice 1975), experts share a level of communication with each-other that is hampered by the interference of non-experts. "Ask Systems" are meant to approximate conversations, and this means that the content analysis must accommodate the kind of interaction the end users would have with the experts in person. The best way to do this is to let representatives of the end-user group point the way.

Indexing questions requires cross-training. Either an indexer must be trained in the domain or an expert must be trained to index. As discussed above, the indexer (as question poser) must be able to think like a member of the end user group. The person posing the questions should, however, have some idea of what is being done with the questions and should have a feel for the kinds of questions that are most useful.

Spontaneous reactions to clips are good because they produce good questions. Because we cannot predict the path users take to get to any clip, it is best to take into account the reaction users are likely to have to a clip without reference to other clips. Indexers are also more likely to pose the question really on their minds spontaneously, while new agendas may take over if they stop to think about it.

Spontaneous reactions to clips can be problematic because they often produce questions not answered elsewhere in the network. This is simply because the library is limited. When a question finds no answer in the network it points to a need for material to be covered in future interviews.

The more specialized the domain, the less call there is for "deep" remindings. The most relevant cases are the most specifically similar to those encountered in the real world by the end user. Consulting for tax accounting needs to come across as being very "concrete", even if the ultimate goal (in the case of TaxOps, increasing thinking across specializations) is subtle. Similarly, in a very specific domain there is little value in "cross-contextual" remindings, even if the concept at play is quite general, like *"Don't forget to follow up"* or *"Don't overlook the details"*. As with "deep" remindings, our users wanted to hear wisdom from their peers.

Acknowledgements

The authors would like to thank Roger C. Schank, Christopher Riesbeck, Ray Bareiss, Richard Osgood, Maureen Zabloudil, Edward Lazar, Michael Huesing, Jorn Barger, Andrew Fano, Robert Hooker, Melanie Wallis, Michael Engber, Tamar Offer-Yehoshua, Andy Gordon, Ian Underwood, Ann Jacobs, Jarret Knyal, Marita Decker, Dennis Gruener, and Wallace Head for their contributions to the design and implementation of TaxOps.

This Research was supported in part by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058 and the Office of Naval Research under contract N00014-90-J-4117, by the Office of Naval Research under contract N00014-89-J-1987, and by the Air Force Office of Scientific Research under contract AFOSR-89-0493. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech, an Institute Partner, and from IBM.

References

Bareiss, R. (1989). *Exemplar-based Knowledge Acquisition: a Unified Approach to Concept Representation, Classification, and Learning*, New York: Harcourt-Brace Johanovitch.

Grice, H.P. (1975). Logic and Conversation, in P. Cole and J. Morgan (Eds.) *Syntax and Semantics 3: Speech Acts*, New York: Academic Press, 41-58.

Riesbeck, C. and Schank, R. C. (1989). Inside Case-Based Reasoning, Hillsdale, NJ: Lawrence Erlbaum Assoc.

Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*, Cambridge, UK: Cambridge University Press.

Schank, R. (1991). Case-Based Teaching: Four Experiences in Educational Software Design, *ILS Technical Report #7*, The Institute for the Learning Sciences, Northwestern University, Evanston, IL.

Schank, R., W. Ferguson, L. Birnbaum, J. Barger, M. Greising (1991). Ask Tom: An Experimental Interface for Video Case Libraries, *ILS Technical Report #10*, The Institute for the Learning Sciences, Northwestern University, Evanston, IL.

GENERIC TECHNIQUES IN EMA: A MODEL-BASED APPROACH FOR KNOWLEDGE ACQUISITION

Susan Spirgi, Dieter Wenger

Swiss Bank Corporation, Hochstrasse 16, 4053 Basel, Switzerland

André R. Probst

IBM Switzerland and University of Lausanne, General Guisan Quai 26, 8048 Zürich, Switzerland

Abstract

The model-based problem solving is a natural extension within the development methodology EMA, because EMA is based on the basic principles: abstraction, decoupling and parallelism. A classification of problem types is made by classifying the solution information of the problems. Model-based problem solving is done by G-TECs (Generic TEChniques). A G-TEC consists of a knowledge model and its realisation and is herewith formal and operational. The knowledge model of a G-TEC and that of an application are based on the same meta-model (modeling language). The use of G-TECs leads to the integration of better problem solving methods in the applications, to a better knowledge acquisition and last but not least to a strong reduction of coding work. Prototype-, meta-model-, result- and model-based knowledge acquisition are brought together to a whole.

1. INTRODUCTION

EMA is a methodology for the development of knowledge-based applications. In this paper the model-based approach for knowledge acquisition is outlined.

Chapter 2 gives a short introduction of the basic concepts and principles of EMA. Chapter 3 introduces the underlying conception of this model-based approach and leads to a classification of problem types. Readers familiar with KADS or other methodologies may read chapter 5 before chapter 4. In chapter 5 EMA is related to other methodologies. In chapter 4 the model-based approach by means of G-TECs is explained in detail. Chapter 6 emphasizes the importance of the developer's ability to work on different levels of abstraction. Chapter 7 concludes this paper.

2. INTRODUCTION OF EMA

EMA [7] is an integrated, active and comprehensive methodology for the development of knowledge based applications. It is integrated because the methodology itself is realized as a knowledge-based application. Due to that it is an active partner for the developer. EMA is comprehensive because information of the whole development cycle are kept and managed and EMA supports the developer in each step of the development cycle even in generation of code (now ART-Code, soon TIRS-Code).

The two most important concepts are the information-set (info-set) or information-type (info-type) and the agent. The concept 'info-set' is based on the concepts 'object', 'object-instance' and 'object-class'. 'Object-instance' and 'object-class' are both objects. An info-set is a set of information-units (info-units). In figure 1 OC-x is a object-class and OI-x1, OI-x2, OI-x3, etc. are object-instances. An info-unit is defined by an object and one attribute. In figure 1 OI-x1@Att-y, OI-x2@Att-y and OI-x3@Att-y are info-units. These info-units can be seen as elements of the info-set OC-x@Att-y.

Sometimes instead of the expression info-set the expression information-type (info-type) is used, because OI-x@Att-y is the type of the info-units OI-x1@Att-y, etc..

An info-set, described by an object and one of its attributes, comprises as elements all info-units with the same attribute as the info-set and with objects which are in an is-a- or in an instance-of-relation to the object of the info-set. Info-sets are info-units. Info-sets comprise info-units and are representatives of a set of info-units.

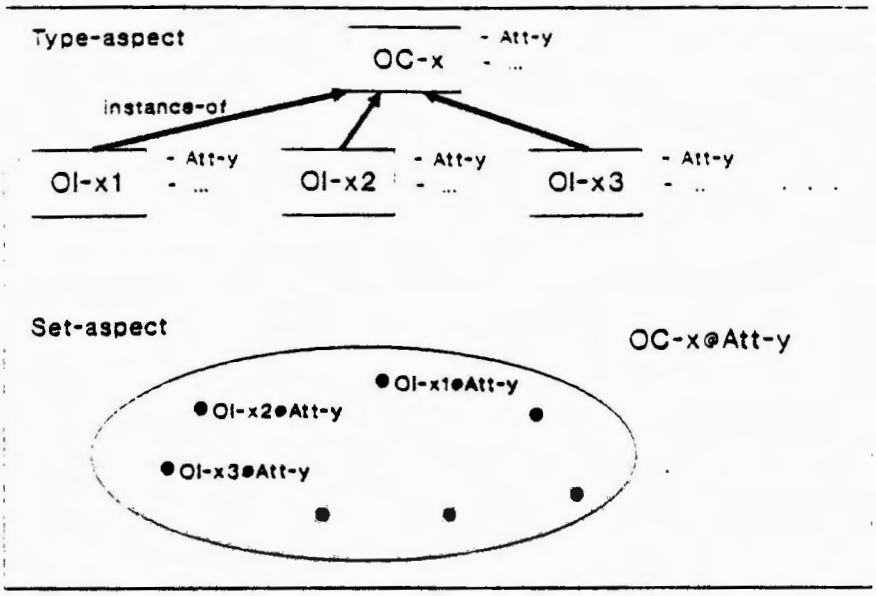
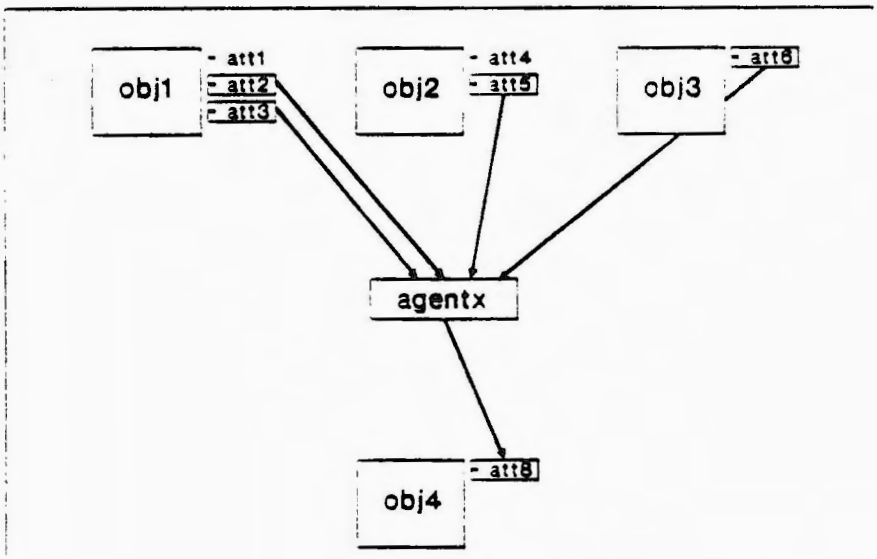


Figure 1 information-set, information-type, information-unit



The concept agent manipulates the info-unit(s) of exactly one info-type (info-set). For this the agent needs some different kinds of information (knowledge), which is expressed by info-sets.

- **activation knowledge:**
  The information specifying when the agent has to become active.
- **input knowledge:**
  The information about input info-sets which have to be processed by the agent for producing the output info-set.
- **constraint knowledge:**
  The information about info-units of the input info-sets which have to be selected.
- **process context knowledge:**
  The information that drives the processing (processing parameters).
- **functional knowledge:**
  The knowledge that is in functional form, specifying how the processing is done. It makes the agent operational.

Due to the fact that an agent manipulates the info-units of exactly one info-type, an agent is related to the object-class of this info-type. The important consequence is that the agent is given the abstraction level of this object-class. Because processing is done by the agents the processing is related to abstraction.

Figure 2 shows an agent that manipulates the info-set obj4@att8. For this it needs info-units of the info-sets obj1@att2, obj1@att3, obj2@att5 and obj3@att6. These info-units belong to the above mentioned kinds of knowledge.

Figure 3 shows how an application is seen. A system is represented in figure 3 where the agents become active depending on some of the info-units and process input info-units to output info-units. The produced output is again taken as input by other agents (black-board principle). In [7] the bottom up support of EMA is described. In the following chapters the top-down support is outlined.

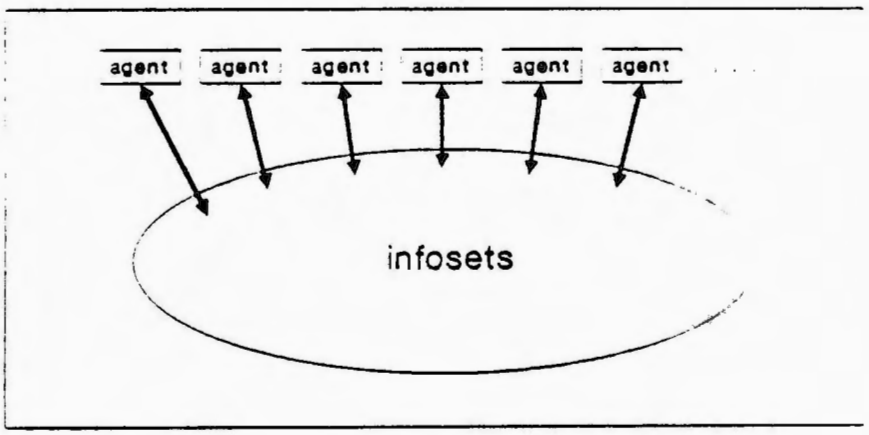


Figure 3 Agents manipulating the info-sets (blackboard principle)

3. PROBLEM SOLVING (PS) IN EMA

The inferencing of information from known information is seen in EMA as PS. Therefore PS may be sometimes trivial sometimes extremely difficult and complex. Figure 4 shows the basic concepts of PS: the problem-description, the PS-procedure and the solution-information (solution-info) of the problem (result of the PS procedure). These basic concepts are related to each other. The problem and the solution-info are related in a 1:1-relationship, whereas the PS-procedure is related in a n:m-relationship to the problem and herewith the solution-info. This means that it is possible that the same solution-info can be built by more than one PS-procedure and that it is even possible that different kinds of problems can be solved by the same PS-procedure.

For making a classification for PS we have to classify at least one of the basic concepts. If we classify the problems, we classify the solution-info and vice versa, because they are related by one to one. It doesn't seem very promising to classify the PS-procedures, because the number is huge and the procedures are hard to describe in a formal way. A formal description facilitates the classification very much.

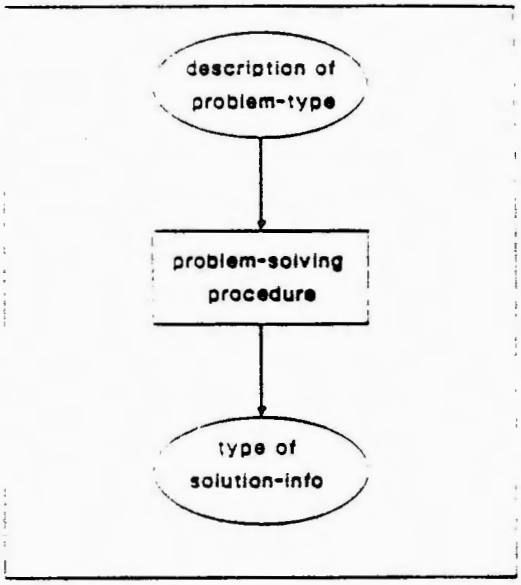


Figure 4 Basic concepts of problem-solving

There are some reasons for classifying the solution-info:

- The solution-info can be entirely expressed with help of info-sets, because it is an information. This means that the things we want to classify can be easily expressed in a formal way. This is an important precondition for classification, because how should it be possible to classify something if it cannot be well described. The solution-info is classified because it is easier to express the model of the solution-info than the model of the problem-description
- For the classification of the solution-info we have a strong criterion: What is the solution-info used for and interpreted as by the user? For this the underlying assumption is that information becomes its meaning and herewith its value as soon as and not before it is used. A strong criterion is another important precondition for classification.

Although the starting conditions for classification are excellent we get into trouble when we analyse and intend to classify the infered informations of different applications. The number of types of information is growing tremendously. Two examples shall be discussed.

There is an information which is interpreted as a ratio of a balance sheet analysis and there is an information which is needed by a user for evaluation (evaluation info). Expressing these two types of information with info-sets gives us:

- balance sheet ratio @ value
- ? @ evaluation

The first info-set, built by the object-class 'balance sheet ratio' and its attribute 'value' (remember an info-set is built by an object-class and one of its attributes), is interpreted as 'the value of a balance sheet ratio' which is considered the class of information. This class of information is limited to a specific domain which is the domain of balance sheet analysis, because the object-class 'balance sheet ratio' is an object-class of this domain. Therefore it can be said that the solution-info 'ratio of balance sheet analysis' is domain-specific.

The second info-set shows a solution-info which is not domain-specific. An infinite number of different objects-classes may have an evaluation. It is not possible to relate these objects to a certain domain. Therefore the solution-info 'evaluation of ...' is a general solution-info.

As a conclusion we get solution-infos (and herewith problem-descriptions) which are general or which are domain-specific. For the classification of solution-infos the concept 'domain' has to be taken into account. So one classification exists, which is domain-independent (general) and one classification for each of the different domains. In other words the huge amount of different kinds of solution-infos can be partitioned into general classes of solution-infos and into the different domain-specific classes of solution-infos.

Figure 5 shows that the domains build an abstraction structure (domain, sub-domain, etc.). Each of these domains have their own classification of solution-infos. This would lead to domain-oriented model-based knowledge acquisition.

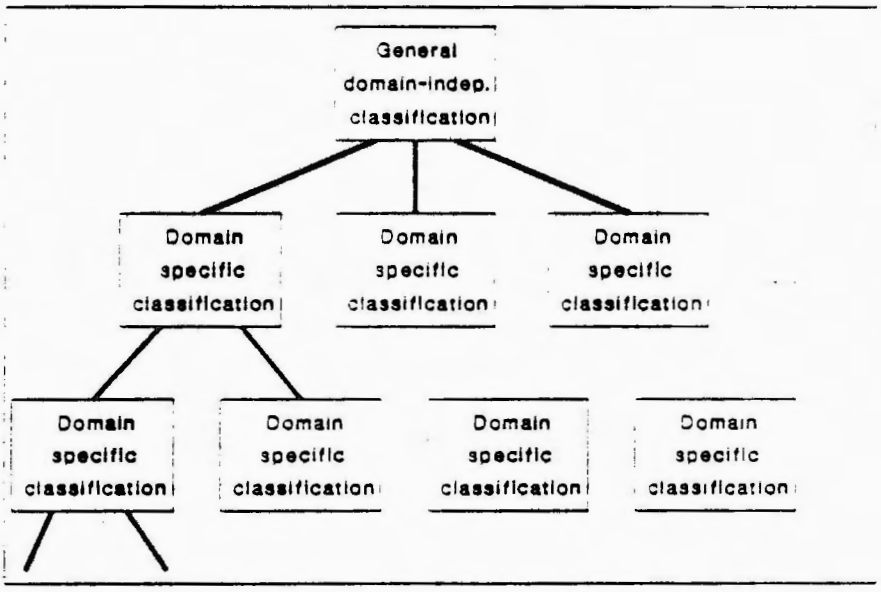


Figure 5 General and domain-specific classification of solution-info

In this paper we restrict ourselves to the classification of general kinds of solution-infos. The current classification which is not yet complete looks like:

- classification info
- prediction info
- evaluation info
- causal info (diagnosis info)
- planning info
- design info
- configuration info
- hint info
- modification-info (advisory info)

Although sometimes prediction info is built by classification a difference is made between 'classification info' and 'prediction info', because our classification of solution-info is made according to its use.

4. GENERIC TECHNIQUES (G-TECs)

In EMA the building block which should solve a certain kind of problem is called G-TEC. Two kinds of G-TECs exist, the conceptual G-TEC (CG-TEC) and the technical one (TG-TEC). The produced result of a CG-TEC can be classified according to the classification given in chapter 3. A CG-TEC can be related to a certain kind of problem (problem-specific). A TG-TEC is problem neutral (e.g. an optimization technique, statistical technique). TG-TECs are often parts of CG-TECs.

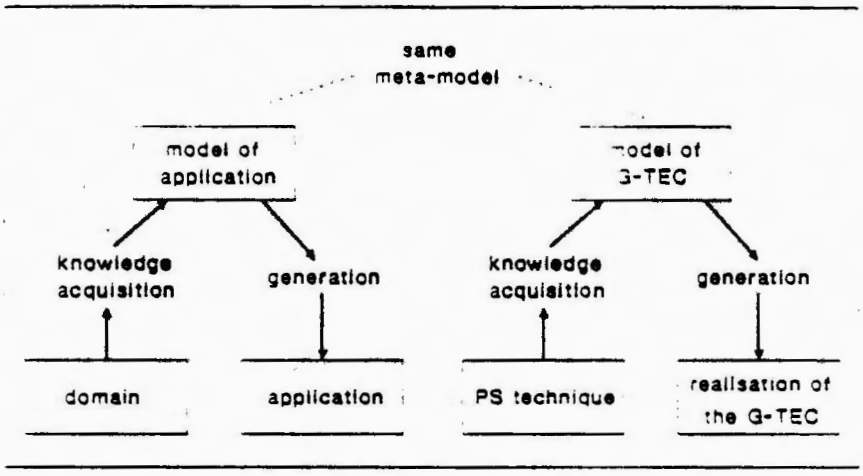


Figure 6 Development of an application and of a G-TEC

4.1. BUILDING A GENERIC TECHNIQUE

The generic technique approach is model-based and has some parallels to generic task [3] [4] and the interpretation model approach in KADS [1] [2].

The developing of G-TECs is principally similar to the developing of applications. At the left hand side of figure 6 the development of an application is shown. The model is made with information which is acquired from the domain. The model is the model of the application and not of the domain. The application is generated with the information of the model (knowledge model).

At the right hand side of figure 6 the development of a G-TEC is shown. The information for the G-TEC model is acquired from a PS-technique which is transformed into a G-TEC. Then the G-TEC is generated with the information of the G-TEC model and some manually provided code which makes the G-TEC operational. A G-TEC consists of its model and its realization (operational part). The models of the G-TECs and the applications are based on the same meta-model (modeling language).

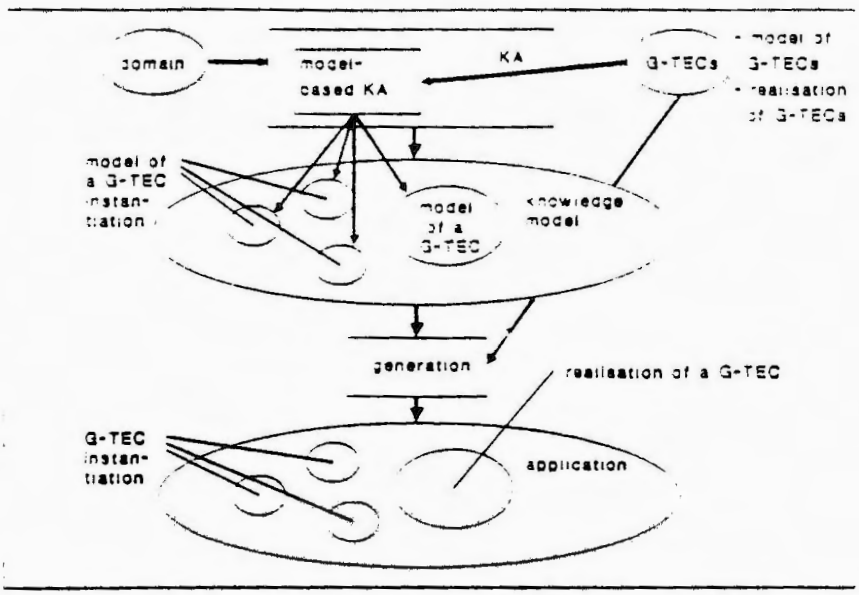


Figure 7 Building an application with G-TECs

4.2. BUILDING AN APPLICATION WITH G-TECs

In EMA besides of model-based knowledge acquisition there are other kinds of knowledge acquisition. From a G-TEC we have its knowledge-model and its realization (operational part). Figure 7 shows how an application is developed with G-TECs. Integrating a G-TEC in an application means to make one ore more instantiation(s) of a G-TEC. The models of the G-TECs and their instantiations become a part of the knowledge model. When the application is generated, the instantiations of the G-TECs are created and integrated in the application. The G-TEC part of an application is obtained without any programming (apart from application-specific methods), because the entire processing of a G-TEC instantiation is inherited from the G-TEC (see also figure 9 and 10).

In connection with the sisyphus project of EKAW91 the development of a whole application is described in [8].

4.3. EXAMPLE OF A CG-TEC: CONDEV

CONDEV is a technique for the condensation of evaluations. A frequently occuring problem is the evaluation of a certain object (a person, company, etc.). It is assumed that information about different partial aspects of this object exists and that this information can be related to evaluations (partial evaluations). The problem is to condense these partial evaluations to intermediate evaluations and to the final evaluation. The mechanism should be robust against incompleteness of information about this object.

The knowledge model of a G-TEC is built with the meta-concepts of EMA. The model of the G-TEC CONDEV can be described with two object-classes, some attributes and two agents (figure 8).

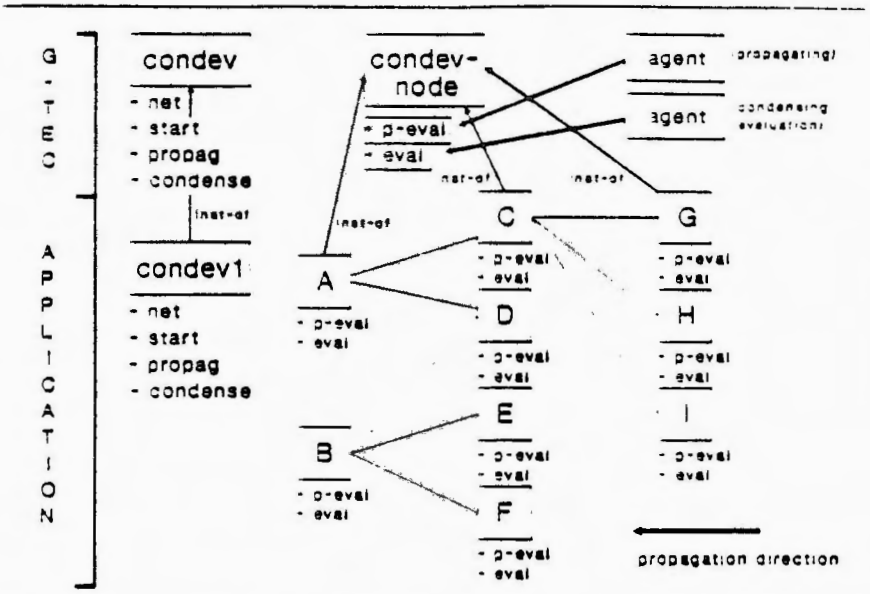


Figure 8 G-TEC CONDEV (CONdensation of Evaluation information)

Figure 8 shows the CG-TEC CONDEV within an application. The upper part of figure 8 is the model of CONDEV and is independent of the application. The lower part is the model of an instantiation of CONDEV and is specific to the application.

The principle of the condensation of evaluation is the following:

Each node of the net (A, B, C, D, E, F, G, H and I are instances of condev-node) has the attributes 'p-eval' and 'eval' which are inherited from the object-class 'condev-node'. The attribute 'p-eval' contains primary and propagated partial evaluations which consist of an average and of a certainty. Primary partial evaluations are the evaluations which have not been created by propagation. They are the input information of CONDEV. There is an agent which propagates the partial evaluations from one node to the node on the next higher level (from the right to the left in figure 7). The second agent of the G-TEC CONDEV builds the evaluation (attribute 'eval') which is based on the partial evaluations of the same node. This evaluation remains local and only the partial evaluations are propagated. The processing is specific to CONDEV which means that the processing is done independent of the net-structure which is specific to the application.

In the CG-TEC CONDEV the attribute 'net' of the object-instances of 'condev' contains the information about the net-structure.

The attributes 'propag' and 'condense' are method-attributes which means their values are method-names (function-names). The method 'propag' is used by the propagating, the method 'condense' by the condensing agent. Some standard methods are provided that differ in the weighting mechanism. The developer selects one of them. In case he needs a different mechanism he can customize a standard method.

4.4. OTHER G-TECs

The number of G-TECs is not limited. All techniques that do any processing may be transformed into G-TECs; e.g. machine learning , reasoning , neural net, operation research techniques, etc.. Very often some techniques can be taken together to a class of G-TECs and classes of G-TECs to more general classes and we get a generalisation structure for the G-TECs. This generalisation can be extremely useful, because some attributes and agents can be attached to object-classes on a higher

level of abstraction; e.g. there are classes of neural net techniques with agents that are specific to the object-class 'neural-net'.

4.5. SETTING UP THE RELATIONSHIP BETWEEN G-TECs AND PROBLEM CLASSIFICATION

Although the G-TECs are modeled and constructed according to their nature (machine learning techniques belong together) we have a different criterion for applying them. We classify according to the kind of produced result. We classify whether the produced information is an evaluation, a planning, a classification information, etc.. The criterion is the following question: What is the produced result used for and interpreted as by the user?

4.6. MODEL-BASED KNOWLEDGE ACQUISITION WITH HELP OF G-TECs

1. A required result is identified in the task-domain and expressed with info-types.
2. The type of the required result is defined with help of the PS classification. The question which should be asked and which represents the criterion for selection is: What is the produced result used for and interpreted as by the user?
3. If at least one CG-TEC exists, knowledge acquisition driven by the underlying CG-TEC and supported by EMA is carried out. The decision whether a CG-TEC is selected depends on the availability of the information, which must be acquired in the task domain, and herewith on the success of the knowledge acquisition process. A successfull knowledge acquisition process leads to a model of the CG-TEC instantiation which may be integrated into the knowledge model of the application (see also figure 10).
4. The selected CG-TEC implies different info-sets. Some of these info-sets have to be produced first. The info-sets become required results. For getting a required result go to 1..

All these steps are carried out cooperatively by developer and EMA.

EMA is able to produce hints to draw the developer's attention to inconsistent situations as e.g. missing input or context information and EMA helps with visualisation features, so e.g. the managing of the net-structure in CONDEV.

It should not be forgotten that very often no CG-TEC exists for a required result. Then the part of the knowledge model for obtaining this required result has to be built from scratch with the help of other kinds of knowledge acquisition provided by EMA [7].

5. POSITIONING OF G-TECs

In this chapter EMA and its G-TEC approach is positioned to other approaches. A comparison is first made to KADS, because KADS is currently a well-known approach. Then a positioning is made following the structure of 'Models, methods, roles and tasks: Many labels - one idea!'[5].

5.1. COMPARISON EMA - KADS

KADS has a **multi-layer conceptual model** and a **design model.** EMA only has a **knowledge model** which comprises the two KADS models. EMA has **views** which correspond roughly to the layers of KADS. Views are special representations of the **knowledge model** and show specific aspects: so the **information view** (IV) shows the information aspect, the **local agent view** (LAV) the processing aspect and the **global agent view** (GAV) the control aspect. The **domain layer** corresponds to the IV (not vice versa), the **inference layer** to the LAV and the **task and strategic layer** to the GAV. The **knowledge source** of KADS corresponds to an **agent** and the **meta-class** would be in EMA expressed by an **object-class**, which has a 'has-role' relation to other object-classes. The **problem solving method** of KADS is called in EMA a **G-TEC.** The **interpretation model** corresponds to the

knowledge model of a G-TEC. A **task** in KADS corresponds to an **activity** in EMA. EMA has also the concept **task** but it is used in a completely different manner. It is the task that is carried out cooperatively by user and application. In EMA only one modeling level exists. The **implementation level** of KADS is the **prototype level** (application) in EMA.

5.1. POSITIONING OF EMA ACCORDING TO [5]

The term model in EMA contains the information about an application or a G-TEC and is therefore something abstract because information is also abstract. Aspects of the model are made visible with help of different views (representations).

The problem solving method is modeled with the same meta-model (modeling language / formalism) as the application, but the model of the problem solving method is on a higher abstraction level, because the object-classes within the model are on a high level of abstraction and therefore also the attached processing, because it obtains the abstraction level of the object-classes (see chapter 2). The G-TEC models are problem-specific and herewith independent of the application. Compared to Generic Task and its interaction problem, where knowledge cannot be represented free of assumption of how it will be used, in EMA the information which is used is based on the same meta-model. Information becomes knowledge as soon as and not before it is used for solving a problem. The same information may be used for solving different problems.

EMA is in opposite to the life cycle principle of KADS, where analysing of knowledge and expertise can and must be done before design and implementation, based on a **knowledge model and parallel prototyping approach** which means that the different parts of an application are at one moment on different levels of concreteness. The levels of concreteness span the range between 'no model at all' to 'operational'. The different parts are developed in parallel but decoupled in the sense of their state. This approach stands also in opposite to rapid prototyping. It demands some qualities of the underlying principles of the methodology which are: decoupling of the processing building blocks, abstraction of the information and of the processing and parallelism. This is achieved by a data-driven black-board based agent and object-oriented approach.

Similar to KADS the knowledge model can be developed from scratch (meta-model-, result-, prototype-driven) and with the help of G-TECs.

The corresponding terms to the terminology developed in [5] are:

- Task in [5]: Task has in EMA a completely different meaning. User and application fulfill a task in cooperation (cooperative problem solving). The term activity corresponds to task in [5]. An activity may be carried out by the user or the application and is caused by an event. An activity is a part of the whole task.
- Problem solving method in [5]: The G-TECs are the problem solving methods in EMA. They consist of a model and an operational part (realisation of the G-TEC).
- Action of the problem solving method in [5]: The agent corresponds to this term.
- Modeling level in [5]: The model of all of the G-TECs are based on the same meta-model (modeling language), as the knowledge model of an application. EMA generates the code of the implementation environment (ART, soon also TIRS). The operational part of a G-TEC, which is problem-specific and herewith independent of the application, consists of code of the implementation environment.
- Conceptual model in [5]: In EMA no distinction between conceptual and design model is made. It is called the knowledge model and contains the whole information-based knowledge about an application. The functional knowledge which makes the application operational is not contained in the knowledge model but attached to it.
- Control knowledge in [5]: Control knowledge is one kind of knowledge used by the agent for doing its job at the right moment within an activity (activity in the sense of EMA).

In [5] three hypothesis have been set up:

Hypothesis 1:

It is useful to describe problem solving behaviour on a more adequate, abstract level than is offered by general purpose knowledge representation languages.

In EMA the problem solving behaviour is described by the model of the G-TEC which is based on the same meta-model (modeling language) as the knowledge model of the application. Moreover a coded part is attached to the model which makes the problem solving operational.

Hypothesis 2:

Models of problem solving behaviour can be specified in a problem specific, but application-task independent way.

Figure 9 gives an overview over model-based problem solving in EMA.

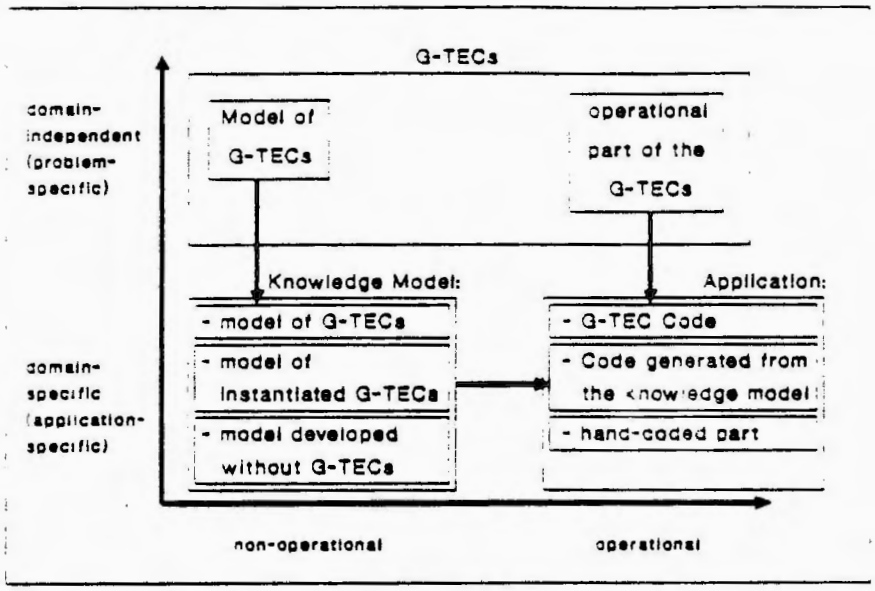


Figure 9 Relationship between G-TEC and Application ordered according to specialisation and operationality

EMA provides a modeling language (meta-model) for the modeling of the G-TECs. Moreover the G-TECs are operational. The different G-TECs can be easily combined, because they are parts of the same knowledge model and based on the same meta-model (modeling language).

For choosing the right problem solving method (CG-TEC) EMA gives a support with the help of a problem taxonomy and of the G-TEC-specific KA facilities: EMA proposes one ore more G-TEC(s) and shows the user the needed information for applying a CG-TEC. The user checks whether the needed information is available in his task domain. If so the CG-TEC may be applied.

Figure 9 shows the model-based problem solving in EMA.

Hypothesis 3:

Models guide the knowledge acquisition process and structure the knowledge base.

The support for knowledge acquisition provided by EMA is described with help of figure 10.

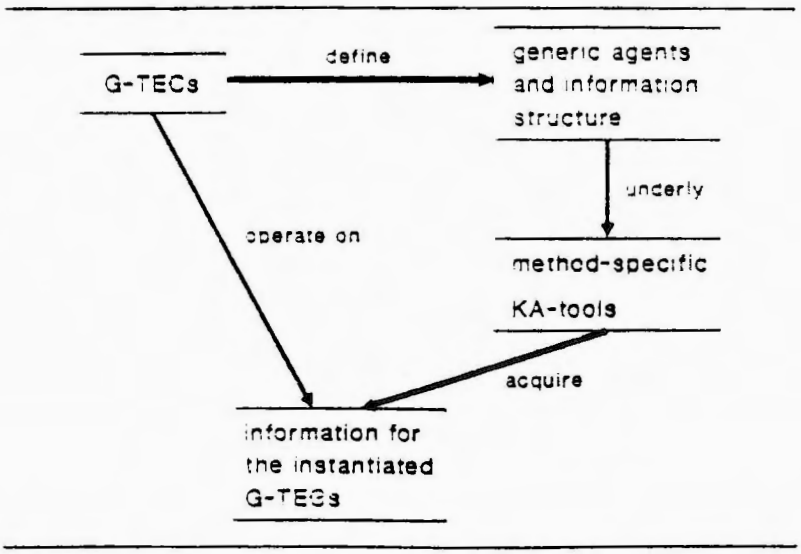


Figure 10 Relationship between G-TEC and Knowledge Acquisition

The knowledge base is structured by the knowledge model, because the information-based knowledge of the knowledge-base is generated from the knowlege model and because the operational knowledge is attached to the information-based knowledge of the knowledge model.

6. LEVELS OF ABSTRACTION FOR THE DEVELOPMENT

With EMA the developer is working on different levels of abstraction.

The most abstract level is the G-TEC-level. The next level is the agent-info-set-level, then the level of the knowledge-based tool (ART or TIRS) and at last the implementation level of the tool (language C).

7. CONCLUDING COMMENTS

The G-TEC approach in EMA is a concrete realisation of a part of the vision that is formulated in [6] about generic models and in [5] about model-based knowledge acquisition.

The G-TEC approach is a natural extension of EMA due to the basic principles of EMA which are decoupling of processing, parallelism and abstraction.

EMA has become an integrated, comprehensive and active methodology which supports the developer from the first to the last step in application development. Nevertheless there is still a lot of work to do: more techniques must be transformed into G-TECs; EMA must be optimized: it must become more efficient and more stable; it must become more developer friendly, etc..

REFERENCES

[1] Breuker J. (ed), Wielinga B., Schreiber G., de Greef P., de Hoog R., van Someren M., Wielemaker J., Billault J.P., Davoodi M., Hayward S., (1987) Model Driven Knowledge Acquisition: Interpretation Models, Del A1, Esprit p1098, University of Amsterdam

[2] Breuker J., Wielinga B., (1989) Models of Expertise in Knowledge Acquisition, Memorandum 103 of the VF-project "Acquisition of Expertise"

[3] Bylander T., Chandrasekaran B., Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition, Int. J. Man-Machine Studies (1987) 26, pp. 231-243

[4] Chandrasekaran B., Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, IEEE Expert (Fall 1986), pp 23-30

[5] Karbach W., Linster M., Voss A., (1990) "Models, methods, roles and tasks: many labels - one idea?", Knowledge Acquisition 2, Academic Press Limited, pp. 279-299

[6] Shadbolt N., Wielinga B., (1990) Knowledge based knowledge acquisition: the next generation of support tools, EKAW-90, Current trends in knowledge acquisition, IOS

[7] Spirgi S., Probst A., Wenger D.,(1990) "Knowledge Acquisition in a methodology for knowledge-based applications", proceedings of JKAW '90, pp. 382-397

[8] Spirgi S., Wenger D., Probst A., Sisyphus Project EKAW-91

MULTIS: A Knowledge Acquisition System Based on Problem-Solving Primitives

Yuri Adrian Tijerino
Tadahiro Kitahashi
Riichiro Mizoguchi

Institute of Scientific and Industrial Research, Osaka University
8-1 Mihogaoka, Ibaraki-shi, Osaka 567 Japan

ABSTRACT

The aim of this project is to develop a method for automating knowledge acquisition that can be employed to automatically generate knowledge-based systems making few initial constraints on the type of problem solving mechanism. This method is based on the following concepts: (1) human problem solving can be broken down into conceptual primitives that can be reused and rearranged to model problem solving processes from vast domains and task types, and (2) there exist enough formal problem solving mechanisms in algorithmic form, or at least those mechanism can be easily developed by computer experts, to implement those models in a machine understandable code. To prove the tractability of these concepts we are currently developing MULTIS (MULti-Task Interview System).

Keywords: Interview system, cognitive primitive, generic vocabulary, generic process, building block, case-based reasoning.

1. INTRODUCTION

In the past two decades intensive research has taken place in the area of knowledge-based systems in numerous countries. As a result, so-called "expert systems" have been constructed to store human problem solving knowledge in the computer in a dynamic form that can be used as an aid to the knowledge provider and other parties interested. Originally, those systems, which will be referred to as knowledge-based systems (KBS) hereof, were computer programs designed by "knowledge engineers," a term coined by Dr. Faigenbaum in the early years of artificial intelligence. Knowledge engineers are usually persons that would spend numerous hours talking to human experts, who in some cases would think of them as intruders poking in their own business. Through this painful but often rewarding period, knowledge engineers have accumulated readily useful knowledge of their own that has been written in the form of papers, books, etc. This knowledge of how to acquire knowledge from experts and encode it into computer programs, can now serve another very important purpose, that is, to create systems based on that knowledge to automate the process of knowledge acquisition.

Interview systems are computer programs that are constructed with that purpose in mind. Early interview systems were generated from existing knowledge-based systems and were useful in aiding in the construction of other knowledge-based systems that presented problem solving mechanisms similar to that

of the original one. Based on this fact, it was proposed that there exist cognitive primitives (Chandrasekaran, 1986; Clancey, 1985; McDermott, 1988, Tijerino et al., 1990) in which terms most human problem solving could be modeled. But how could that be possible for all facets of problems? Problem solving knowledge was subdivided in two major areas: task knowledge and domain knowledge. From this perspective, primitives could then be identified for task knowledge. Since it was also proposed by the above researchers that task knowledge presented strong guidance in the process of acquisition of domain knowledge, then those primitives would be of great help in that process too.

The aim of this project is to develop an interview system, MULTIS, based on the concept of cognitive primitives, also known as mediating representations. The major difference between this project and other related ongoing projects is that we make a distinction between domain expert's cognitive primitives and knowledge engineering primitives. Other projects consider the only the latter leaving it up to the knowledge engineer to analyze the domain expert's problem solving process and synthesize KBSs with reusable mechanisms. We propose that the former can be extremely useful in the process of modeling problem solving interactively with the domain expert and that the former can be mapped to the latter, thus automatically generating the target KBS.

2. TWO-LEVEL DOMAIN EXPERT'S COGNITIVE PRIMITIVES

We make no claim that we have identified "the" primitives of human thought. On the contrary, in this paper an experimental framework is presented in which conceptual primitives do not necessarily have to be precise in modeling human thought, but only accurate in creating systems that model thought processes in the form of computer programs.

In this section, we will present two types of cognitive primitives used to model expert problem solving processes: generic vocabulary and generic processes. Notice that these are only equivalent to domain expert's cognitive primitives and no attempt is made to implement them directly as computer programs. As figure 2-1 illustrates, their implementation is accomplished by mapping them to mechanisms that represent knowledge engineer's primitives which we refer to as building blocks and will introduce in

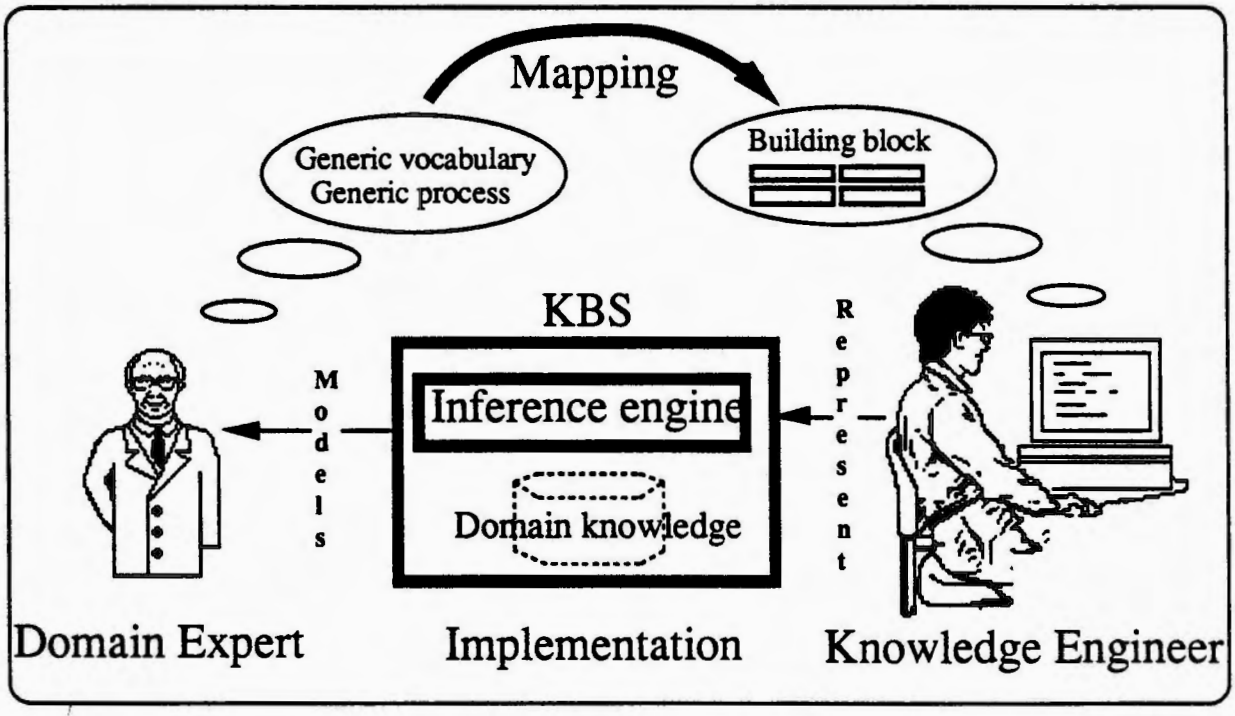


Figure 2-1. Implementation of a KBS is accomplished by mapping domain expert's primitves to knowledge engineer's primitives.

another section. A comparison of our research with other ongoing research efforts will also be given. Last, we will provide an example of how an already existent system can be modeled with those primitives.

2.1 Generic Vocabulary

One kind of cognitive primitive used by all fellow human beings can be identified as the vocabulary. Not a particular language itself, e.g. English, Japanese or Spanish, but the concept represented by the vocabulary of the language, e.g. the concept given by the word "book." It is important to make this distinction because it is what makes communication possible or impossible between two parties. For instance, let's suppose a person A does not know the meaning of the word "book." Let's suppose further that another person B utters the word "book" to A in A's language. Tough A will not understand the word, it does not imply A's ignorance of the word itself, e.g., by hearing the word A has just learned of its existence. On the other hand, A will not learn the meaning of the word, i.e. the concept implied by the word, until either B shows a book or explains its meaning to A. Furthermore, once the assimilation process has been completed, A will add the new concept given by the word to his/her vocabulary, not his/her language, and will be able to use it or understand it in future occasions.

Vocabulary can either be very generalized or very specialized. Generalized vocabulary is learned at the period of adaptation during one's youth and as one becomes older or experienced at doing some particular task or job one learns more specialized vocabulary. The latter is usually explained in terms of the former, which is not a reciprocal relation. For instance, the concept of book can be explained in terms of words such as paper, writing, tales, drawing, etc., but those words cannot be explained in terms of book without major effort. As such vocabulary becomes more and more specialized, it becomes dependent on some particular task and common only to individuals with experience in that task. Furthermore, as one gains a vocabulary related to an specific task in which one is experienced, e.g. medical or equipment failure diagnosis, that vocabulary becomes the basis for explaining concepts more specialized and for communicating with colleagues, in common task-dependent vocabulary, without resorting to more generalized terms. This is the kind of vocabulary to which we will refer to as "generic vocabulary." Generic because it is common to domain experts performing the same task type, but not necessarily in the same domain.

We assume that there exist this kind task-dependent terminology for each PS task that can be represented in the form of generic vocabulary. Generic vocabulary can serve two purposes, that is, to analyze the tasks for hidden cognitive primitives, which we call generic processes* and to help the interviewer be fluent and coherent during interview.

In the real world, generic vocabulary can be found in most fields of human cognitive processes. Let's revisit our word-processing example. In most, if not all, word-processing programs one can use the words mark, delete, cut, copy, paste and others. Nevertheless, the name of this actions (verbs) may vary from implementation to implementation, not to mention from language to language. No matter how different one may choose to call these verbs, the action intended is still very similar if not the same. Due to this fact, when a person becomes well acquainted with one word processor it becomes very simple for him/her to use any other word processor once the necessary mapping or translation of the previous word-processor's vocabulary has been made to the new word-processor's vocabulary.

In this early stages of our project, we have identified a generic vocabulary for the task of scheduling. In table 2-1, the vocabulary we have identified for scheduling is depicted. The words in italics represent verbs or actions which we also call active vocabulary. The rest are called passive vocabulary because usually actions are performed on them. The passive vocabulary is divided into two types: those with strong task dependence, represented in outline font, and those with weak task dependence, represented in normal font.

Table 2-1. Experimental generic vocabulary for scheduling.

analyze	*delete*	*relax*
apply	*generate*	requirement
assign	*identify*	resource
batch	job	*satisfy*
capacity	*merge*	*select*
categorize	*modify*	sequence
category	preference	schedule
conflict	priority	subschedule
constraint	process	tolerance
.		*update*

Italics : action Normal: weak task-dependence

Outline: strong task dependencce

2.2 Generic Processes

Generic processes are another kind of mediating representation between the domain expert's cognitive primitives and knowledge engineering primitives. Their main difference with generic vocabulary is that generic vocabulary are mostly partial concepts with no significant meaning unless combined with others and generic processes are primitive concepts of activities described in terms of generic vocabulary. Generic vocabulary are passive in the sense that though some of them represent an action, they must be combined with other vocabulary to completely describe the action, e.g. what is the object of the action. As the term mediating representation implies, generic processes are not "the" cognitive primitives of human thought and no attempt is made to categorize them as such. The generic processes presented here for scheduling are experimental processes that have been changing since first introduced in order to better and more generally represent the processes undertaken in scheduling. They stand some where in the spectrum between real domain expert's cognitive primitives and knowledge engineer's building blocks.

Generic processes are primitives defined in terms of generic vocabulary. In our word-processing example, a generic process could be paragraph-deletion for which, for instance, a paragraph has to be marked before being deleted. The generic process to generic vocabulary relation can be written as: paragraph-deletion = delete marked paragraph. The most important relation between generic vocabulary and generic processes is that generic processes usually combine two or more generic words to describe a general action, not another word.

We have identified some generic processes for planning and scheduling, but intend to expand to other PS tasks such as monitoring, diagnosis and so forth. Although, perfect generic vocabulary and generic process libraries cannot be constructed, we believe that they provide enough guidance for analysis and synthesis of the tasks under consideration. What matters the most, is that the mechanisms, to which the generic processes were mapped, are able to successfully accomplish their final goal, that is, that they will help construct (synthesize) the desired KBS. In MULTIS, a mapping between generic processes and knowledge engineering mechanisms is made in such a way that one or more generic processes may correspond to a single mechanism.

Figure 2-2 presents some generic processes identified for scheduling and simple planning problems. We have divided the generic processes in two levels: generic processes and generic sub-processes. The

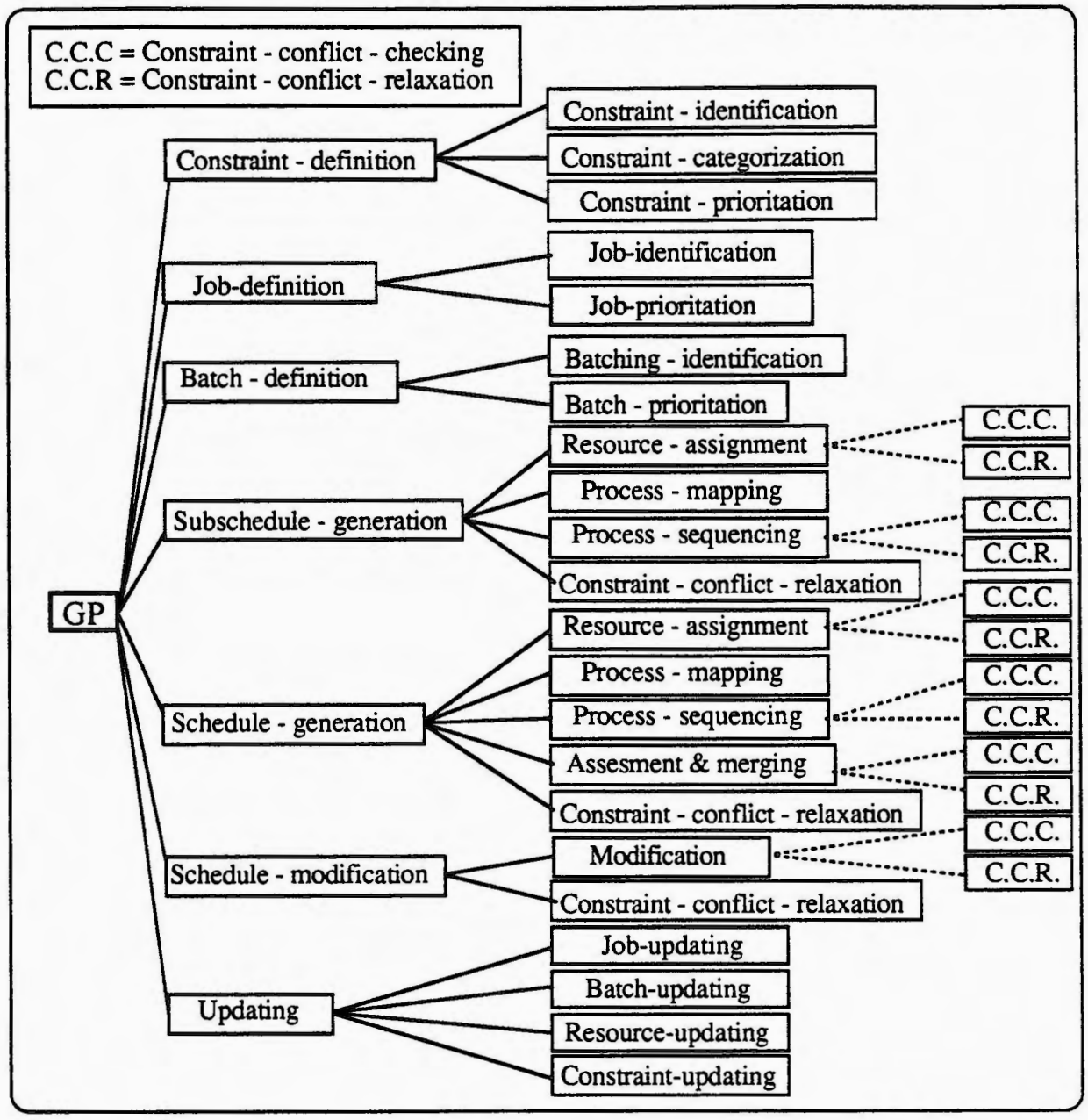


Figure 2-2. Experimental generic processes for scheduling.

former are more specific to the task and the later more general, that is, the later are more general and can be used to explain the process of the former. In this figure, the left column depicts the generic processes and the right columns the generic sub-processes. The lines between columns indicate modularity or specialization. Appendix A presents a more detailed explanation for each of the processes shown in figure 2-2.

Generic processes can be interconnected in what we refer to as generic process networks. The connectivity between generic processes is restricted by the kind of conceptual input or output, represented in the form of generic vocabulary, that they can receive or send respectively. Thanks to this connectivity relation, inference strategies can be created to aid in the process of identifying what generic processes a particular domain expert employs when performing a specific task.

2.3 Related Work

What follows is an enumeration of some related research efforts being undertaken in the area of cognitive primitives.

2.3.1 Chandrasekaran's Generic Tasks. Chandrasekaran (1986), also Bylander and Chandrasekaran (1987), proposes units of search control mechanisms that are very useful to construct inference engines for KBSs and to guide in the elicitation of domain knowledge. However, those units called "generic tasks" or "building blocks," though generalized for a specific type of PS tasks, are too specialized in the context of knowledge engineering. Therefore, they are very difficult to be understood by the domain expert himself.

Comparison: Our generic processes differ from Chandrasekaran's generic tasks in that our generic processes have to be combined in order to obtain knowledge-based system building blocks. In other words, our generic processes are used to model human PS at the cognitive level and, if properly translated, can be easily understood by the domain expert. Chandrasekaran's generic tasks can be used as building blocks as they are.

2.3.2 Clancey's heuristic classification. Clancey (1985) proposes heuristic classification as a method that is useful to explain the mechanism of most KBS systems constructed in the area of analysis tasks. Again, this type of method is easily understood by knowledge engineers, but not by domain experts.

Comparison: In MULTIS we attack the problem of generalization of problem solving methods from quite a different perspective from that of heuristic classification. Heuristic classification, though very successful a method, is too general and doesn't provide enough insights for knowledge acquisition. It is difficult to believe that, because problems usually involve more than one type of task, all analysis problems can be best modeled with only heuristic classification. In MULTIS, the general assumption is made that more than just one problem solving method can be used to represent human problem solving processes in computer programs. However, if Heuristic classification is broken up in smaller component pieces, it would still be possible to provide generic processes that could be mapped to those component pieces, thus behaving as mediating representations.

2.3.3 McDermott's Half Weak Methods. Introduction of half-weak methods by John McDermott (1988) contributed greatly to knowledge engineers in building KBSs. However, as Chandrasekaran's generic tasks, it is difficult for an average domain expert to understand his/her own PS process in terms of such half-weak methods.

Comparison: Again, these so-called half-weak methods are similar to MULTIS's building blocks, because they are helpful in modeling of inference engines for knowledge based system construction. Conceptually, half-weak methods are better employed to explain the way computers solve problems than the way humans do. However, as it is the case with heuristic classification, we believe that generic processes and generic vocabulary can be found to stand as mediating representations between the domain expert's cognitive primitives and McDermott's half-weak methods.

2.3.4 Weilinga's Knowledge Sources. Breuker and Weilinga (1989) propose to make models of human PS and call the units used in those models "knowledge sources." This is an interesting idea because those knowledge sources seem to be somewhere in between what we have called generic vocabulary and generic processes. Therefore, though KADS, the system that implements their proposal, is intended for knowledge engineers, its potential for usability by domain experts seems to be high.

Comparison: KADS's knowledge modeling framework uses generic processes and generic vocabularies (though not called that way) to model human problem solving. Although this is extremely similar to the MULTIS's modeling, KADS doesn't remember models nor it tries to adapt those models to other domain experts and was intended as a knowledge engineer's tool.

2.4 Discussion

As noticed in previous subsections, our idea of cognitive primitives depart somewhat from the context of knowledge engineering. However, we believe that cognitive primitives proposed by other researchers are very useful as implementation tools or units for KBSs. The cognitive primitives proposed in this paper are intermediaries between human cognitive primitives and computer search control mechanism. Therefore, they provide a way to overcome the representation mismatch between domain expert and knowledge engineer, not human and machine.

2.5 A hypothetical example

Let's suppose that expert A is an expert at performing a scheduling task in domain X. Let's further suppose that there is another expert B who also is an expert at performing a scheduling task, but in domain Y. Let's also assume that X is different from Y significantly. Lastly, let C be a knowledge engineer with the task of constructing two KBSs, both for modeling A's and B's problem solving knowledge in scheduling for domains X and Y respectively. Since A and B perform a common task in different domains, their task dependent primitives might also be common. Therefore, if those primitives are translated accordingly, A and B can understand each other's problem solving processes. C, on the other hand, will have to study much deeper about each of A's and B's domains and their relation to scheduling before endeavoring to model KBSs for them respectively. In other words, C has to map his/her primitives (building blocks) to A and B and learn about X and Y in order to construct useful KBSs. Once KBSs for doing scheduling in X and Y have been modeled, their task-dependent mechanisms could be very similar, because they were modeled using C's primitives (building blocks).

In this example, A and B are able to understand each other after some comparatively short discussion of X and Y. This is true because their task is the same, thus, they possess common primitives for performing it. Most of the time spent at discussing would probably be used in translating vocabulary related to X or Y. On the contrary, C has to do more effort because he has to learn new primitives (generic vocabulary and generic processes) and/or map them to his own primitives (building blocks) that are not from scheduling, but from computer programming. However, as time goes by and C constructs more scheduling KBSs, he will become more and more acquainted with such mappings and consequently more efficient at building scheduling KBSs.

3. BUILDING BLOCKS

Building blocks are transparent to the interviewee, but not to knowledge engineers who usually use them to design and construct a KBS. Though MULTIS builds a model of PS interactively with the interviewee using generic vocabulary and generic processes, a mapping of such model is made transparently to the building blocks needed to construct the inference engine of the knowledge-based system intended. This approach is consistent with the knowledge engineer's approach because they code the inference engine without showing it to the interviewee after building his/her PS model interactively.

At the end of the previous section, we compared our mediating representations with other research projects. It was concluded that the representations we introduced could still be used as mediating representations between domain expert's primitives and the primitives (building blocks) proposed by those projects. However, in this section we will introduce some experimental KBS building blocks used to prototype KBSs that perform scheduling tasks.

3.1 Summary of Mechanisms

As the name building block implies, they are mechanisms used to implement KBSs. In figure 3-1, we present some experimental building blocks. These building blocks are intended to be as general as possible in representing problem solving mechanisms identified by knowledge engineers through the years. Their combination will result in an implemented prototype for the inference engine of a KBS. Though mechanisms proposed by others (Chandrasekaran, 1986 and 1987; Clancey, 1985; McDermott, 1988) could be used as building blocks also, we have adopted this approach for simplicity

reasons. Thus, we don't make any claim that the building blocks presented here cover all facets of knowledge engineering problem solving primitives. On the contrary, we propose that building blocks presented by other researchers be also used.

In figure 3-1, building blocks are divided into two types: primitive building blocks and composite building blocks. The former are more general than the later which are derived as combinations of the former. Again, the same argument introduced previously, that there exist general primitives and more specific primitives, is still valid because as it can be seen, the primitive building blocks are general knowledge engineering concepts, while the combinational building blocks are more specific to given situations. We could also argue that there is almost no difference between the level of generic processes and this building blocks, since both endeavor to represent human cognitive primitives. However, there exist some relevant features that make distinction between them possible, that is, generic processes can be differentiated from building blocks on the following two accounts:

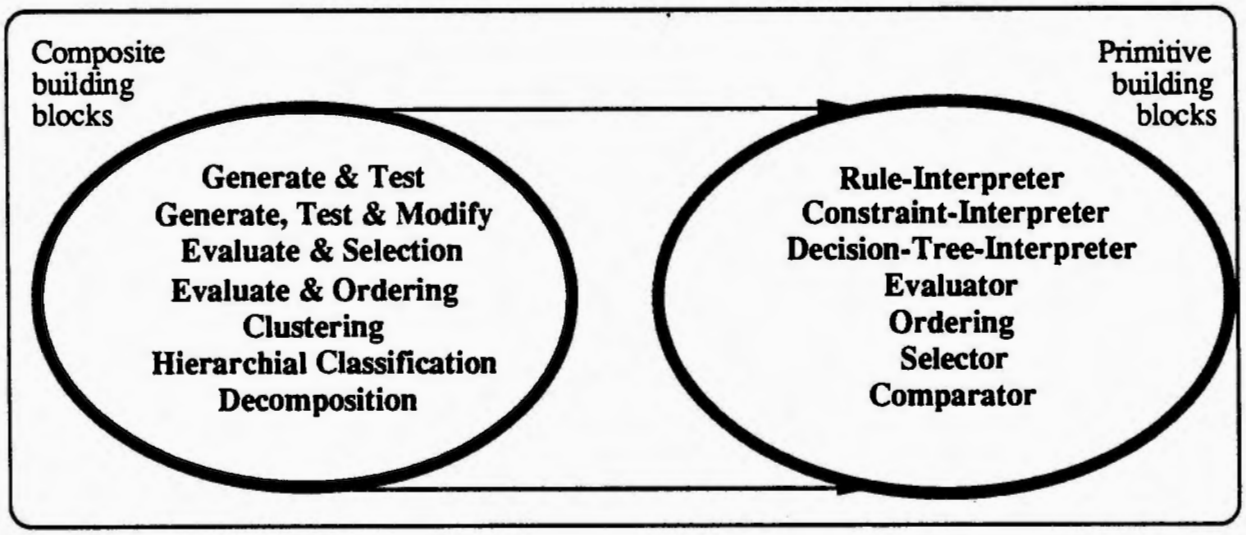


Figure 3-1. Experimental building blocks divided in composite and primitive ones.

1) There exist no consensus in any domain or task on what generic processes could be used to successfully model problem solving for that domain or task. On the contrary, building blocks have been proposed so far that can be used to generally model knowledge engineering problem solving.

2) Generic processes mainly model domain expert problem solving in a representation that is understandable to them, but cannot be used to explicitly implement a executable computer program. On the other hand, building blocks are representations understandable by the knowledge engineer and can be used to construct computer programs, i.e. KBSs, that implement their problem solving models.

Appendix B, gives a non-exhaustive enumeration of experimental, combinational building blocks being proposed in this paper.

3.2 Mapping of Generic Processes to Building Blocks

The mapping of generic processes to building blocks requires expertise of knowledge engineers in doing such mapping. Though not formally recognized, knowledge engineers have engaged in the task of doing such mappings when confronted with construction of KBSs. The knowledge engineer interviews domain experts, who can generally only express their problem solving in their own way, and then identifies what could be used as implementing blocks also in a rather personal manner. Consequently, a mapping is implicitly established between the domain expert's cognitive primitives and the knowledge engineers library of implementing mechanisms. The expertise used in creating this mappings can be extracted from

knowledge engineers and used to perform mappings between networks of generic processes and building blocks proposed in this paper.

In appendix B, there is some guidance as to what kind of knowledge can be used for mapping generic processes to building blocks. For each generic process presented, there is an entry by the name of "mechanism identification inference," from which interview strategies could be generated to identify building blocks useful to implement the domain expert's problem solving model.

3.3 An Example

CABPRO (Schaefer et al., 1988) is a simple KBS developed to schedule processes when assembling cables. A cable could be very complex or very simple to make and very few cables could be considered standard, thus, the processes for making them vary in great degree. A schedule for making a cable is a very detailed set of instructions called a "traveler," because the person making the cable must travel from working station to station to complete the cable according to the instructions given in the traveler. Although a traveler is considered as a schedule, it must be taken into consideration that there are many travelers at any time in the factory, hence, a traveler is only a sub-schedule to the whole scheduling problem. For this reason, a traveler will be referred to as a sub-schedule.

Figure 3-2a shows what generic processes can be used to model CABPRO's task of scheduling. In figure3-2b, a translation to the CABPRO's vocabulary is given. Figure 3-2c illustrates the mapping established between the generic processes and the building blocks used to construct the inference engine of CABPRO's prototype.

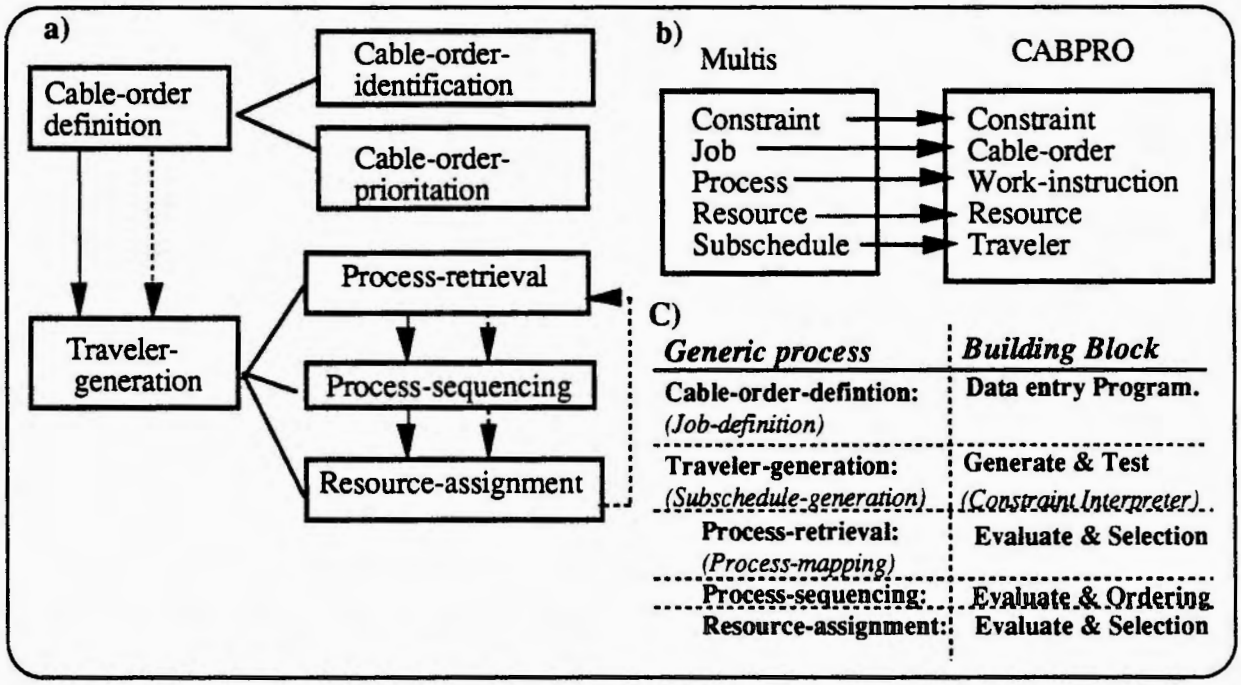


Figure 3-2. CABPRO's a) generic process network, b) generic vocabulary translation, and c) generic process to building block mapping.

4. MULTIS

We are currently developing an interview system based on the previous considerations that domain expert's primitives can be mapped automatically to knowledge engineer's primitives. What follows is a brief description of the system.

4.1 Interview Steps

MULTIS's interview strategy consists of the following four phases which are described below:

1. Task type identification. Identify the PS task the expert performs.

2. Task analysis and synthesis. Analyze the task and identify needed search control knowledge.

3. Knowledge acquisition. Elicit domain knowledge and task implementation knowledge.

4. System evaluation. Evaluate the system by imputing an already solved problem.

4.1.1 Task Type Identification. MULTIS first identifies the task the domain expert, from whom the knowledge is to be elicited, performs during PS. During this phase of the interview, the tasks for which MULTIS can elicit knowledge are currently simple planning and scheduling, but can be expanded later to cover other tasks such as diagnosis, monitoring and so on. Since a taxonomy of tasks already exists for the type of tasks being performed by humans , this phase can be implemented as a classification problem. Therefore, the interview is based on heuristics about the taxonomy of tasks. Roget (Bennett, 1985) is an earlier interview system that attempts to identify the type of task the interviewee performs before engaging in knowledge acquisition. However, it limits the type of task from which it can elicit knowledge to diagnosis by using sentence-parsing techniques that could prove somewhat inefficient if other tasks were to be added. MULTIS performs interactive interview with a user friendly interface to avoid falling into a similar problem.

Since an expert sometimes performs more than a single task in his/her field of expertise, it is necessary to identify what is the main task being performed. Once that task is identified, the rest of the tasks being performed can be ignored, because later during the interview their corresponding generic processes will be incrementally added to the model of the main task. This is a problem that hasn't been treated properly in the past because most interview systems have assumed that human experts perform only a single task. However, human PS is usually more complex, that is, it consists of a mixture of analysis and synthesis at different levels. For instance, a medical doctor mainly performs diagnosis (analysis), but he also designs treatments (synthesis) to fit special cases of patients. With this in mind, MULTIS identifies the main task the expert performs and then incrementally identifies what other tasks are being performed.

MULTIS then extracts the generic vocabulary corresponding to the task identified and translates it to the interviewee's vocabulary. As the term generic implies, a generic vocabulary is generalization of a vocabulary for a given task. Therefore, a vocabulary must be translated to reflect the interviewee's vocabulary. In MULTIS the vocabulary is translated interactively by the user himself.

4.1.2 Analysis and Synthesis. Next step is to analyze and synthesize the task identified in the previous phase. MULTIS makes use of the generic vocabulary that corresponds to the task identified in the previous step to determine what type of generic processes can be related to it. Then, with the generic processes identified heuristics are employed to single out building blocks that can be used as part of the search-control structure for the target KBS through case-based reasoning.

MULTIS makes use of case-based reasoning during its interview to reuse previous combinations of generic tasks encountered in past interviews to make search for generic tasks more efficient. The vocabulary corresponding to the expert's task and its translation are employed as indexes to search in

the case base for a similar case or cases. Later, the generic task combination from the extracted case is modified with use of a graphical interface (under development) to accurately represent the task being performed by the domain expert. Finally, the generated generic task combination is analyzed heuristically to identify the right mix of building blocks that should be used as part of the search-control structure of the target KBS. The details of case-based reasoning are given in another paper (Tijerino, 1990).

PS modeling and inference engine mapping take place in this phase. Interactive modeling is implemented through the graphical interface mentioned after a prestored modifiable model has been found. The inference engine mapping is performed transparently after employing heuristics to identify building blocks useful to reflect the patterns given by the model.

4.1.3. Acquisition of Domain Knowledge. In this step, the required additional task-implementation knowledge and domain knowledge is elicited from the domain expert. For this purpose, the assumption is made that each generic process and each building block, related to the task involved, have structures that guide the processes of elicitation of domain knowledge, and task-implementation knowledge respectively. Chandrasekaran (1987) and McDermott (1988) have made similar assumptions. Put in other terms, a strategy for knowledge acquisition can be directly mapped to each generic task or each building block.

As explained before, the main characteristic of MULTIS is its ability to help acquire knowledge for various tasks. Therefore, we have concentrated mostly in the design of inference engines for those tasks through building block combination. However, acquisition of domain knowledge is also a very important issue that cannot be ignored. Consequently, we are carrying out intensive research on the mapping of existing knowledge acquisition methods to the search-control structures resulting from MULTIS's interview.

4.1.4 System Evaluation. The last phase in the interview strategy, is system evaluation. At this point, MULTIS tries to solve a problem already solved by the expert being interviewed. If the expert is satisfied with the solution then no more questions are asked and the target KBS is constructed. Otherwise, MULTIS checks for missing or erroneous domain knowledge, or erroneous search control knowledge.

4.2 Architecture

MULTIS architecture is depicted in figure 4-1. It consists of the following modules and submodules:

1) Task Type Identification Module, which interactively identifies what is the main task type that the domain expert perform during problem solving. The purpose of this module is to help MULTIS focus on a task type to avoid unnecessary search for PS primitives from other task types

2) Generic Vocabulary Translator, which also interactively helps the domain expert translate his/her vocabulary to make the rest of the interview flow coherent with the interviewee.

3) Case-based Reasoning Module. This module uses the translation of generic vocabulary and patterns found in it to search for previous case which present some similarity with the current. Next, it interactively modifies the generic process network of the similar case with help of the Generic Process Simulator to reflect the PS structure of the interviewee. Then, it maps the generic process network obtained to implemented building blocks to yield the inference engine of the target KBS.

3.1) Generic Process Simulator. This is an user-friendly module that takes the interviewee through tutoring sessions on generic processes, shows examples of similar cases' generic process networks and simulates the flow of those networks during PS. It also provides the interviewee with powerful means to modify those networks in such a way that he/she can represent his/her own PS structure in a new generic process network.

3.2) Generic-Process-to-Building-Block Mapping Module. This module uses heuristics obtained from knowledge engineers to map generic process networks to building blocks yielding an executable inference engine.

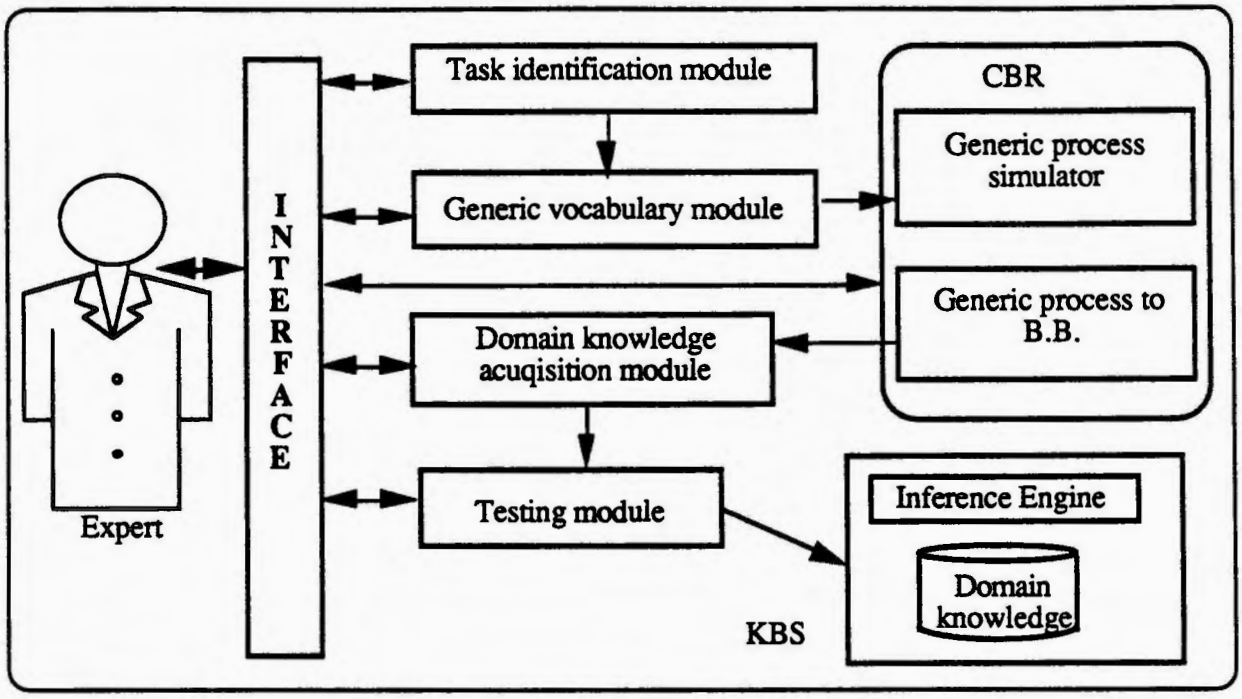


Figure 4-1. Multis Architecture.

4) Domain Knowledge Acquisition Module. This module employs interview strategies attached to building blocks and generic processes present in the current case.

5) Testing Module. This module attempts to solve an already solved case as explained in 4.1.4.

Details of each sub-module will be presented in future papers.

5. Conclusion

MULTIS is still under implementation. Once fully constructed it should be able to:

1. Engage in coherent and smooth interview with domain experts with help of generic vocabulary.
2. Model the PS processes of domain experts in various fields in the form of generic process networks.
3. Identify which prefabricated search control structures can be implemented to construct a KBS that performs the desired problem-solving.
4. Make use of strategies related to the generic processes and building blocks to elicit domain and task-implementation knowledge.
5. Help construct or quickly prototype KBSs for various tasks.
6. Make use of CBR to improve efficiency during the interview.

Currently, MULTIS possesses a case base for process scheduling with seven cases. We plan to add other cases that cover other tasks such as planning, monitoring and diagnosis. Also, mapping between generic tasks and building blocks, a very important issue, is being investigated along with the corresponding mapping to domain knowledge acquisition methods.

REFERENCES

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition.* Chandrasekaran, B. (ed) Academic Press.

Bennett J. S. (1985) ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System, *J. of Automated Reasoning* I, pp. 49-74.

Breuker, J and Wielinga, B. (1989) Models of expertise in knowledge acquisition. *In topics in Expert System Design*, G Guida and C Tasso, eds. 265-295.

Bylander, T and Chandrasekaran B. (1987) Generic tasks for knowledge-based reasoning: the "right" level of abstraction for knowledge acquisition. *Int. J. Man-Machine Studies.* 26, 231-243.

Chandrasekaran, B. (1986) Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert* 1, 23-30.

Clancey, W. J. (1985) Heuristic Classification. *Artificial Intelligence* 27, 289-350.

Kahn, G., Nowlan, S. and McDermott, J. (1985) MORE; an intelligent knowledge acquisition tool. *Proceedings of the 9th Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 581-584.

McDermott J. (1988) Using problem-solving methods to impose structure on knowledge. *IEEE International Workshop on Artificial Intelligence for Industrial Applications.* 7-11.

McDermott, J. et al. (1990) Explorations in how to make application programming easier. *In Porceeding of 1st Japanese Knowledge Acquisition fo Knowledge-Based Systems Workshop.* 134-147.

Numao, M. and Morishita S. (1988) Scheplan—A scheduling expert for steel-making process. *In Proceedings of IEEE International Workshp on Artificial Intelligence for Industrial Applications*, 457-472.

Schaefer, R. M., Colmer, J. S. and Miley, M. (1988) CABPRO: A rule-based expert system for process planning of assembled multi-wire cables. *IEEE* CH2552-8/88 (181-186).

Tijerino, Y. A., Kitahashi, T. and Mizoguchi, R. (1990) A task-analysis interview system that uses a problem solving model. *In Porceeding of 1st Japanese Knowledge Acquisition fo Knowledge-Based Systems Workshop.* 331-344.

Appendix A: Generic Processes for Scheduling

What follows is an enumeration of the generic processes and sub-generic processes presented in figure 2-2. The order is given in the same manner as they appear in figure 2-2. Each generic process will be described according to the following description criteria:

1) a definition in terms of generic vocabulary,

2) input and output taken,

3) a general explanation of the concept they describe,

(0) *Constraint-definition*

Definition: Identify and prioritize constraints neede for scheduling.

Input: Constraints from the world.

Output: Constraints.

Explanation: With this generic task the individual performing scheduling defines what are the constraints needed in performing problem solving for scheduling.

(0.1) *Constraint-identification*

Definition: Identify constraints needed for scheduling.

Input: Constraints from the world.

Output: Constraints.

Explanation: Identify the relevant constraints that will be considered during scheduling.

(0.2) *Constraint-prioritation*

Definition: Prioritize constraint.

Input: Constraints.

Output: Contraints and priorities.

Explanation: Give an order of importance to each constraint.

(1) *Job-definition*

Definition: identify and/or assign priorities to jobs to be scheduled.

Input: Varies according to what is considered a job, but usually it is the orders taken from the customers.

Output: Jobs, Job priorities

Explanation: Job definition is a process that is performed mostly before the actual problem solving task of scheduling takes place. Although this is a beginning process, it doesn't have to be performed necessarily by the expert himself. This process deals with defining what is to be considered a job, what are the jobs to be scheduled and in some cases what are the priorities to be assigned to those jobs.

(1.1) *Job-identification*

Definition: Identify jobs to be scheduled.

Input: same as (1).

Output: Jobs.

Explanation: Job definition is a process that is performed mostly before the actual problem solving task of scheduling takes place. Although this a beginning process, it doesn't have to be performed necessarily by the expert himself. This process deals with defining what is to be considered a job and what are the jobs to be scheduled.

(1.2) *Job-prioritation*

Definition: Assign priorities to jobs.

Input: jobs

Output: jobs, job priorities

Explanation: Prioritation of jobs can be as simple as determining the priority by a single attribute of the job or as complex as considering multiple attributes as well as other parameters external to the jobs. In essence, this process deals with assigning priorities to jobs before trying to do any scheduling.

(2) *Batch-definition*

Definition: Generate batches, with or without priorities, from jobs.

Input: Jobs, job priorities.

Output: Batches, batch priorities.

Explanation: Jobs may sometimes have similar processes that, if performed simultaneously, permit more efficient and inexpensive factory operation. In this cases, the grouping of jobs into batches is indispensable for scheduling. This process deals with generating groups or sets of jobs that can be treated as units in scheduling. A job priority is one of the many attributes that a batch might posses. However, it is a very important one when it comes to scheduling because jobs with higher priority give a starting point for scheduling. If all jobs are prioritized then there is an implicit partial order from where a schedule can be generated.

(2.1) *Batch-identification*

Definition: Generate batches from jobs.

Input: Jobs, job priorities.

Output: Batches.

Explanation: A batch is a group of jobs with one or more attributes in common. This process uses some grouping criteria to decide what jobs should be grouped. Such criteria might be the combination of various job attributes or a single one.

(2.2) *Batch-prioritation*

Definition: assign priorities to each batch.

Input: Batches.

Output: Batches, batch priorities.

Explanation: A batch priority is one of the many attributes that a batch might posses. However, it is a very important one when it comes to scheduling because batches with higher priority give a starting point for scheduling. If all batches are prioritized then there is an implicit partial order from where a schedule can be generated.

(3) *Subschedule-generation*

Definition: Generate subschedules that satisfy batch or job requirements and/or constraints.

Input: Jobs, Job priorities, Batches, Batch priorities.

Output: Subschedules.

Explanation: This process deals with generating partial, modular subschedules that satisfy local constraints to jobs or batches. Each subschedule should give a series of steps to perform the subschedule locally and in some cases identify what resources have to be used. For some scheduling problems scheduling finishes with this process, that is, by implementing a number of modular schedules problem solving is accomplished.

(3.1) *Process-mapping*

Definition: Select or generate and assign a process to a job or batch.

Input: Job, Batch, processes

Output: Processes assigned to jobs or batches.

Explanation: Process-mapping can be a very simple process, such as assigning a single process to a job or batch, or a very complex one, such as assigning numerous processes in an intricate manner. Processes can also be very simple, being composed of a small set of procedures, or very complex, a large set of procedures. For this reason process-mapping might depend on data-bases where the information about processes is prestored reflecting previous experiences gained during problem solving.

(3.2) *Process-sequencing*

Definition: Generate a sequence of processes for a job or batch.

Input: Processes, Jobs, Batches.

Output: Sequence (partial subschedule).

Explanation: Sequencing is where most of the actual scheduling takes place, because a set of ordered instructions is given in the form of a sequence. Process-sequencing deals with determining an order in which processes must be performed, where processes themselves are predetermined sets of instructions or procedures.

(3.3) *Resource-assignment*

Definition: Select and assign resources to processes, jobs, batches in sequences.

Input: Processes, Jobs, Batches, Sequences.

Output: Processes, jobs, batches in sequences with resources assigned (tentative subschedule or schedule).

Explanation: The main difference between scheduling and planning lies in that in planning the only resource considered is that of time. In scheduling the successful assignment of resources determines the overall success of the task. Resource assignment deals with choosing what resources and in what amount can be assign to a particular job, batch, processes in a sequence.

(3.4) *Constraint-conflict-checking*

Definition: Identify conflict between constraints

Input: constraints

Output: constraint conflict

Explanation: This process deals with identifying what can be treated as a conflict between the various constraints taken into consideration in problem solving, e.g. job, batch, process, resources or other global constraints. This is a process that involves some effort on the part of the problem solver to find which constraints are in conflict with others.

(3.5) *Constraint-conflict-relaxation*

Definition: Relax conflicts between constraints in within bounds assigned.

Input: Constraints in conflict, priorities.

Output: Constraints.

Explanation: When constraints are found to be in conflict with each other the problem solver must decide whether those constraints are flexible, what priorities they have and so on. Then, he/she must decide how to resolve the conflict, e.g. by satisfying only the ones with higher priority, changing the requirements in within the bounds of the constraints, etc. Constraint-conflict-relaxation is the process that is involved in making those changes so as to resolve those conflicts.

(4) *Schedule-generation*

Definition: Generate an overall schedule from jobs, or batches or merge subschedules into a single schedule.

Input: Jobs, Batches, Subschedules.

Output: Schedule.

Explanation: Schedule generation might seem too general a name for this process, however, it is intended to mean the process undertaken when the final schedule is generated from either merging subschedules or constructing one from jobs or batches. Hence this can be a process that yields the final or tentative schedule from subschedules or from either scheduling jobs or batches.

(4.1) *Assessment-and-merging*

Definition: Generate a schedule by merging subschedules.

Input: Subschedules.

Output: Schedule.

Explanation: This process deals with merging subschedules into an overall schedule of the problem being solved (the scheduled being generated). Assessment of how subschedules can be merged forms great part of this process. For most cases, modification of subschedules is performed so that they can be merged without conflicting with each other.

(4.2) *Resource-assignment*

Same as (3.3).

4.3) *Constraint-conflict-checking*

Same as (3.4)

(4.4) *Constraint-conflict-relaxation*

Same as (3.5)

(5) *Schedule-modification*

Definition: Modify a schedule into one or more schedules.

Input: Schedule, constraints, others.

Output: Schedule

Explanation: This process can be as complex as generating parallel schedules during problem solving or as simple as modifying the schedules in a small way at the end of problem solving.

(5.1) *Modification*

Definition: Modify schedule

Input: Schedule.

Output: Schedules.

Explanation: This is a more general process of modification because it serves as the modification engine itself of (5).

(6) *Updating*

Definition: Delete, modify or add jobs, batches, resources, constraints.

Input: Jobs, batches, resources, constraints.

Output: Jobs, batches, resources, constraints.

Explanation: This process deals with deleting , modifying or adding jobs, batches, resources or constraints, during or after problem solving.

(6.1) *Job-updating*

Definition: Delete, modify or add jobs.

Input: Jobs.

Output: Jobs.

Explanation: Same as (6), but only for jobs.

(6.2) *Batch-updating*

Definition: Delete, modify or add batches.

Input: Batches.

Output: Batches.

Explanation: Same as (6), but only for jobs.

(6.3) *Resource-updating*

Definition: Delete, modify or add resources.

Input: Resources.

Output: Resources.

Explanation: Same as (6), but only for resources.

(6.4) *Constraint-updating*

Definition: Delete, modify or add constraints.

Input: Constraints.

Output: Constraints.

Explanation: Same as (6), but only for constraints.

Appendix B: Experimental Building Blocks

The following is an explanation of the building blocks presented in figure 3-1.

Generate & Test.

Mechanism: Generate a schedule according to some given criteria and test the schedule for appropriateness. If the schedule is appropriate, then give it as an output to the problem. If the schedule is not appropriate, then start from the generation process all over again. There are at least three ways of implementation of generators: Selection from a finite set, computation of formulas and production rule interpretation.

Possible primitive building blocks: Rule-Interpreter or Constraint-Interpreter.

Generate, Test & Modify.

Mechanism: Generate a schedule according to some given criteria and test the schedule for appropriateness. If the schedule is appropriate, then give it as an output to the problem. This time if the schedule is not appropriate, then modify it until it satisfies some given conditions or until it can not be modified more, in which case a new schedule is generated and the process is started all over again.

Possible primitive building blocks: Rule-Interpreter or Constraint Interpreter.

Evaluate & selection.

Mechanism: Evaluate a set of known elements according to some given criteria and select the appropriate schedule that covers prestablished conditions.

Possible primitive building blocks: Evaluator, which can be designed with other primitive building blocks such as Rule-Interpreter or Constraint-Interpreter, and Selector which can also be constructed from others.

Evaluate & Ordering.

Mechanism: Evaluate a set of elements according to some given criteria and return a set with the same elements in an order that satisfies a set of prestablished conditions.

Possible primitive building blocks: Evaluator, which can be designed with other primitive building blocks such as Rule-Interpreter or Constraint-Interpreter, and Orderer which can also be constructed from others.

Hierarchical Classification.

Mechanism: Identify to what leaf of a decision-tree an element with some given features can be classified as belonging to.

Possible primitive building blocks: Decision-Tree-Interpreter.

Decomposition.

Mechanism: Decompose a problem into smaller problems until the subproblems become such that they can be used as subschedules. Then, select the most appropriate subschedules that can be employed as the overall solution to the starting problem.

Possible primitive building blocks: Rule-Interpreter

Clustering.

Mechanism: Make groupings of elements that belong to the same category by comparing their features according to some given criteria.

Possible primitive building blocks: Comparator which can be constructed from such building blocks as Rule-Interpreter and Constraint-Interpreter.

Consensus Knowledge Acquisition

Andrew Trice

Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, British Columbia, V6T 1Z2 Canada (Andy_Trice@mtsg.ubc.ca)

Randall Davis

MIT Artificial Intelligence Lab, 545 Technology Sq., Cambridge, MA 02139 USA (davis@ai.mit.edu)

We have developed a method and prototype program, called CARTER, that assists two experts in agreeing on what knowledge should go into a single, consensus knowledge base. Examples of the system in operation are provided, assisting with the reconciliation of two independently developed knowledge-based systems. A significant part of CARTER's task lies in finding the places where the two knowledge bases match, where they differ, and where they overlap. The program displays an interesting degree of success in using circumstantial evidence in the knowledge bases themselves to help infer the degree of overlap in the intended meaning of concepts.

1. INTRODUCTION

At a time when the view is widely shared that valuable knowledge is often distributed among multiple experts, common practice in knowledge acquisition still focuses on acquiring the knowledge of a single individual. At best, system builders may work with multiple experts by appointing one of them as "knowledge czar" and giving him the final word in any dispute, or they may simply require the experts to achieve consensus on their own, without providing any substantive assistance.

Multi-expert acquisition techniques that have been proposed to date have significant shortcomings. Work based on mathematical formulations (Gaglio et al., 1983) tends to be very restrictive; adaptations of established group decision-making techniques (Jagannathan and Elmaghraby, 1985) offer little insight on the issue of consensus; methods based on the repertory grid notion (Boose, 1986; Plaza et al., 1987; Gaines and Shaw, 1989) provide only very basic assistance in detecting potential inconsistencies.

Our research is focused on developing ideas and tools to facilitate the process by which multiple experts construct a single, consensus knowledge base, the process of consensus knowledge acquisition (CKA). We have developed a system called CARTER that is capable of aiding pairs

of experts in this process, by systematically finding, explaining, and suggesting resolutions for discrepancies in their knowledge.

2. HOW CAN WE HANDLE MULTIPLE EXPERTS?

The problem of reconciling multiple points of view has been an issue of study for some time in a number of different areas. Unlike previous work that has focused either on combining decision outcomes (Miner, 1984; Aczel and Saaty, 1983) or supporting the process of logical argumentation (Lowe, 1985, Stefik, 1987; Toulmin, 1958), our concentration is on getting agreement on the knowledge itself.

We do not want to focus on outcome alone, because we believe that the fundamental task is to reach consensus on the knowledge itself: differences in outcome may simply be symptoms of a disagreement about what to know. We choose not to focus on argumentation in the belief that the knowledge representation in use provides an appropriate framework for structuring the discussion. Instead we seek to assist the experts in detecting, deliberating over and reconciling discrepancies in their knowledge. Our approach is normative and focused on the underlying knowledge used by each expert: we want to understand how experts ought to come to agreement and we want that agreement to be about the thing we consider to be fundamental to this undertaking---the knowledge used to make the decisions.

3. BOUNDING THE PROBLEM

Several assumptions help bound the task. First, we assume the expertise to be reconciled is homogeneous in the sense that both experts are capable of solving the entire problem. This enables us to focus on resolving discrepancies rather than combining knowledge from disparate fields.

Second, we assume the experts have constructed individual knowledge bases (KBs) prior to the start of the process. This ensures that the experts can explain the reasoning they used to arrive at their answers and that that reasoning can be adequately captured by a known reasoning process. This in turn allows us to focus on knowledge debugging rather than knowledge acquisition.

This assumption presents some limits: knowledge bases are difficult enough to build that duplicates are infrequently encountered. They do, however, occur, hence the assumption is plausible. The assumption also gives us a place to start work on a fundamentally difficult problem. In addition, as we have speculated (Trice, 1990) and other work has demonstrated (Mark, 1980), the process of knowledge acquisition with a single expert can itself be viewed as establishing consensus: in this case consensus between the expert's view of the domain and the view currently embodied in the knowledge base.

Third, we assume that the experts already have some shared frame of reference, some basic set of terms and assumptions in common. Without that, determining where they agree and disagree would be difficult, not only for our system, but for any human attempting the task. Note, however, that we do not know in advance which terms those are; we presume only that there is

some overlap, up to us to discover. Note, too, that the only information available to our system about the meaning of a term is to be found in the knowledge bases themselves; the system must use that information to infer whether two terms were intended to mean the same thing.

Finally, a hypothesis fundamental to our approach is that a consensus KB can perform better than an individual expert's KB. Our belief is that unearthing and resolving differences between two experts will be fundamentally synergistic, removing limitations and defects in both of their knowledge. This is plausible but of course not guaranteed: some consensus knowledge bases may not be as good as either of the originals.

4. CARTER[1]

Our prototype system plays the role of a non-binding arbitrator mediating between two experts, each of whom has constructed a knowledge base (Fig.1). CARTER examines each knowledge base, looking for matches and discrepancies between them, decides which discrepancy to try to resolve, and suggests possible resolutions. The two experts discuss the suggested resolutions and can choose to update their individual KBs as suggested, update them in some other manner, or not update them at all.[2] The individual KBs are then analyzed anew, until no new discrepancies can be found.

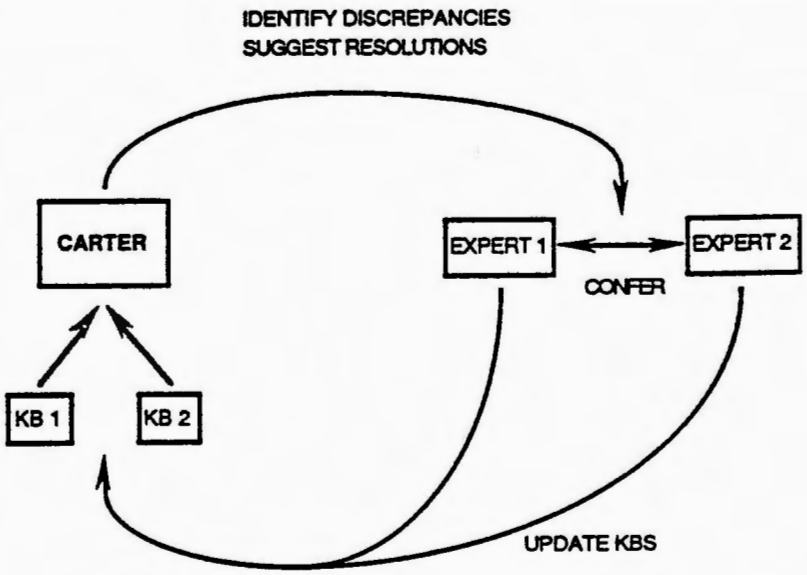


Fig. 1. CARTER Scenario

As one of its first tests, we used the program to assist two statisticians (Bruce and John) in reconciling two knowledge-based application systems they had independently developed. They

[1]Conflict AnalyzeR for Targeted Resolution

[2]That is, we allow them to "agree to disagree."

had each developed a rule-based system designed to assist novice statisticians in detecting and repairing problems in applying linear regression models.[3] While the modeling process involves straightforward arithmetic formulae, there is considerable expertise in knowing the conditions on use of those formulae, determining when the data do not satisfy those conditions, and selecting possible corrective actions. The statistician's assistant system built by each expert captured some part of this expertise. The systems test for validity of the assumptions of linear regressions (e.g., the independent variables are not correlated) to prevent spurious interpretations of data, and they suggest alternatives for improving the model (e.g., model transforms, removing an outlying point from the dataset).

Bruce's knowledge base contained 54 rules, John's 32. The rule language was essentially identical to MYCIN, with rules constructed from attribute, object, value triples (e.g., if the `Defect` of the `Model` is `Multicollinearity...`), plus a degree of certainty. The attributes, objects, and values are the primitive vocabulary of each system; Bruce's knowledge base had a total of 108 such terms, John's had 70.

As a MYCIN equivalent, the rule language was of the complexity used to build many real-world application systems and was sufficiently complex to provide an interesting challenge when attempting to match concepts. With a few dozen rules, the knowledge bases were of modest size, but each was substantive enough to provide good advice about a non-trivial domain. Hence the underlying application systems were modest but real examples of knowledge based systems.

Rule based systems were chosen as a testbed for several reasons. First, the wide use of rules meant that real examples of knowledge bases were available, providing realistic data about the kinds of discrepancies that arise.[4] Second, wide use of rules also means that the specific techniques developed in this work could be widely applicable. Finally, since (as we demonstrate below) the fundamental issue of matching representations arises everywhere, all of the difficult problems arise in working with rules, even though they are a relatively modest representation mechanism.

In analyzing the two knowledge bases CARTER was able to identify 49 out of the total of 76 of discrepancies that existed between them. When working with the experts to help them reach agreement, CARTER was able to assist them in repairing 15 of these discrepancies; this lower figure was determined primarily by the time available from the experts. We illustrate the system's knowledge and performance with examples drawn from this application, simplified slightly for the sake of presentation.

4.1. Carter's Task and Knowledge

CARTER's fundamental task is to determine how and where the knowledge bases match and where they differ. While the knowledge bases may be meaningful to the experts, to CARTER of course they are simply elaborate directed graphs. The system's task is interesting precisely

[3]Linear regressions model the relation between two variables as a linear function and indicate how well the model fits the (possibly noisy) data.

[4]John's knowledge base had been built as an MS thesis in 1986; we provided Bruce with a high level description of the problem solved by John's system and asked Bruce to independently create a knowledge base on the same topic.

because a purely syntactic match of the graphs is of relatively little value, hence this is not a standard graph matching problem. To provide useful assistance CARTER must be able to infer that two concepts that happen to be named differently in the two KBs might in fact mean the same thing.

Strictly speaking, the task is impossible for CARTER to accomplish alone, because only the experts themselves can know for sure what they actually meant. The best CARTER can hope to do is make educated and informed guesses. The system approaches the task with this mindset, trying to find places of most plausible match and most likely discrepancy, then presenting the human experts with its findings and asking for confirmation. The system's utility is thus in asking a small set of well chosen questions, presenting the two experts with the right agenda of things to discuss (possible discrepancies and potential repairs), then making it easy to effect those repairs. The system's effectiveness is determined by how well it can focus that discussion down from the complete space (the crossproduct of concepts from the two KBs) to a smaller, more informed set of choices.

To do this CARTER relies on three forms of knowledge: a knowledge base about the kinds of discrepancies that can arise, a strategy for attacking those discrepancies in a particular order, and a body of circumstantial evidence it accumulates to infer whether two apparently different concepts in fact mean the same thing.

Cataloging Discrepancy Knowledge CARTER's knowledge is organized in a catalog currently containing 35 entries, each of which in turn consists of a discrepancy detection procedure and a corresponding resolution procedure. This simple detection-resolution organization of the catalog facilitates adding new entries as we gain more experience with the task. Examples of each discrepancy type can be found in the Appendix.

Among the types of discrepancies that CARTER can recognize and repair are:

1. Differences in the *character of the result*: one system is content to classify the problem (e.g., specify the nature of a defect in the data, like `Heteroscedasticity`), while the other both classifies the problem and then goes on to suggest what to do about it (e.g., do a `Log-Transform`).

2. Differences in *vocabularies*: one expert refers to the `Defects` of a regression model, while the other refers to its `Problems`, but they are referring to the same thing. Other forms of vocabulary discrepancy the system knows about include differences in representation choice (e.g., one expert represents a concept as an attribute, while the other represents it as a value) and missing terms (e.g., one KB contains values missing from the other).

3. Differences in *pattern of inference*: the experts agree on the overall vocabulary, but interconnect the terms differently, as in the case where one expert uses only an `F-test` statistic to judge the quality of a model, while the other relies on both an `F-test` and an `S-squared` statistic.

4. Differences in the *rules*: the experts agree on the vocabulary and pattern of interconnection between terms, but write different rules. For instance, one expert has a rule that an `F-test` result below a specified threshold indicates that the `Quality` of the `Model` is `Poor`, while the other reasons from the same evidence that the `Quality` of the `Model` is `Fair`. Both reason from the value of the `F-test` to the `Quality` of the `Model`, but they use different rules. Another form of rule discrepancy occurs when two otherwise identical rules have differing levels of certainty.

Ordering Strategy CARTER's overall strategy is to attack these in the order given, beginning by trying to establish agreement on the nature of the outcome. This is motivated by both the computational and negotiation character of the task. Since the computational task is one of matching two KBs that are directed graphs, any useful guidance about where to start the matching process will vastly improve the system's chances of making intelligent suggestions. Expressed in these terms, we anchor the search at the end(s) of the graph, trying first to match the outcomes, relying on the heuristic that two KBs about the same topic are likely to have the same concept(s) as their goal(s).[5] This also makes sense from the negotiation point of view: it is difficult to imagine an effective discussion about the details if the two knowledge bases are trying to arrive at different kinds of conclusions.

Establishing a discrepancy (or correspondence) in the nature of the outcome(s) typically requires dealing with the second task, discrepancies in vocabulary, because the experts may simply have named the goal(s) differently.

The process then works backward a level at a time, attempting to match the nodes (the vocabulary) directly connected to the outcome. Once this is accomplished the system can focus on the patterns of inference that link these nodes with the goal. This in turn enables comparison of the specific rules that establish the goal in each KB.

The process continues from there, moving back one level at a time through the knowledge base, attempting to establish correspondence. The system is also opportunistic in the expected sense, working from any islands of agreement it discovers.

This overall approach, working from goal node to vocabulary, to topology, to specific rules, has proved to be a powerful way to focus the attention of both our system and the human experts working with it, on a task that might otherwise present an overwhelming amount of detail.

4.2. An Example

The examples below show CARTER's first few steps in finding and resolving discrepancies in the two KBs, and illustrates 3 of the 35 types of discrepancies it can detect and resolve.[6]

---

[5] If the KBs have more than one outcome, there is an n^2 problem in matching the outcomes, but this n involves only a very few of the total set of concepts in the KBs.

[6] The program has no interesting natural language ability; all text is generated from simple templates.

Matching Vocabularies: Circumstantial Evidence of Semantic Match In the example at hand, CARTER identifies the goal of Bruce's KB as `Solution` and John's as `Defect`. The concepts are named differently, but perhaps the two experts meant the same thing and simply happened to choose different names? CARTER's most fundamental task is to determine whether this is the case. To do this it accumulates four kinds of circumstantial evidence, available from the knowledge base itself:

- Are the concept labels the same? In this case they are not (`Solution` vs. `Defect`), but this can of course be an artifact of name choice or (in other circumstances) variations in spelling or abbreviation. Conversely, a match in labels is useful evidence but no guarantee of match in meaning.

- In the case of attributes, are they associated with the same object? In this case the answer is no (`Solution` is an attribute of the object `Model`, while `Defect` is an attribute of `Regression`), which is reasonably strong evidence of mismatch (a positive answer is considered only weak evidence of match).

- In the case of attributes, are the values associated with them the same? Here the answer is no (e.g., `Solution` has as values terms like `Log-transform`, `Deflate-by-x`, etc., while `Defect` has as values terms like `Heteroscedasticity`, `Multicollinearity` etc.).

- Are they inferred from the same concepts and are they in turn used to infer the same concepts? That is, do the occupy similar places in the local topology of the knowledge base?[7] Again in this case the answer is no, accumulating substantial circumstantial evidence that Solution and `Defect` are labels for genuinely distinct concepts.

Weighing the evidence in the case at hand, CARTER concludes (correctly) that `Solution` and `Defect` are different concepts, hence there is a difference in the character of the result provided by each system. This matching process is conceptually quite simple, but, as we argue below, an unavoidable part of the task. CARTER's expertise is in the 35 varieties of discrepancies and repairs it knows, not the simple matching process that accumulates evidence of the sort noted above.

The system next attempts to classify this discrepancy in more detail. As noted, CARTER knows that one of the ways in which this discrepancy can occur is if the reasoning chain in one system is longer than the other, as for example if one system only classifies the problem, while the other both classifies it and goes on to suggest a solution. To determine whether this is the case CARTER searches along the reasoning chain in each KB to see whether it can find the goal attribute of one KB on the route to the goal attribute in the other KB. That is, since the two endpoints in the graphs do not match, perhaps the endpoint in one matches one of the intermediate points (conclusions) in the other (Fig. 2).

[7]Here "local" means exactly one step away from the concept in either direction, both because of the weakness of the evidence provided by more distant matches and the expense of complete graph matching.

CARTER tries to match Solution (the endpoint of Bruce's KB) with the intermediate conclusions in John's KB, using the same criteria of label, value, object, and topological correspondence. When this fails, CARTER attempts it the other way around, trying to match Defect of John's KB with the attributes on the route to Solution in Bruce's KB. This second version succeeds when CARTER finds evidence that Defect seems to mean the same thing as Problem in Bruce's KB. Although they are named differently, they are attributes that happen to share two different values (Heteroscedasticity and Multicollinearity), and they are in turn inferred from attributes that themselves appear to match (Determinant and IV-Determinant). This is substantial evidence that these two differently named concepts---Problem and Defect---are in fact identical.

```
I'm afraid these goals SOLUTION and DEFECT do not match.
Also, BRUCE's goal doesn't match up with any concepts in JOHN's KB.

However, JOHN'S goal does seem to correspond to an attribute
determining BRUCE's goal, in particular, PROBLEM, because
     The domain of PROBLEM of BRUCE's KB and
     the domain of DEFECT of JOHN's KB match up:
     Values in common are:  HETEROSCEDASTICITY MULTICOLLINEARITY
     Values only BRUCE has are:  none
     Values only JOHN  has are:  none
     In addition,
     DETERMINANT of BRUCE's KB, which determines PROBLEM, matches with
     IV-DETERMINANT of JOHN's KB, which determines DEFECT.
Therefore, it would seem that SOLUTION is the result of an extra
operation on the data that BRUCE's KB performs while JOHN's KB does
not.
```

Fig. 2: Matching goals

This apparent match enables CARTER to propose a specific diagnosis about the discrepancy between the two KBs. Since the goal attribute of John's KB (Defect), seems to match a concept on the route to the goal of Bruce's KB (Problem), CARTER concludes that Solution is a concept that reflects an additional inference that only Bruce's KB performs (indicated schematically in Fig. 3).

John's Knowledge Base: ... → ... → DEFECT

Bruce's Knowledge Base: ... → ... → PROBLEM → SOLUTION

Fig. 3: Aligning the two KBs.

The generic problem now facing CARTER is that one KB contains a term missing from the other. In response the system offers the experts the obvious choice: either add the missing attribute to the first KB or delete the extra attribute from the second (Fig. 4)

```
I recommend that you do the following to reach agreement on the
goals of the consensus KB:
     First, decide whether SOLUTION should be included in the
     consensus KB.
     If so, JOHN adds the SOLUTION attribute to his KB and generates
     a new set of rules for determining SOLUTION from DEFECT.
     IF not, BRUCE deletes the SOLUTION attribute from his KB, and we
     consolidate the attributes PROBLEM and DEFECT.
Do you approve of this plan? Y
```

Fig. 4: The repair plan.

At this point, CARTER asks the experts to decide. What the outcome, there is additional work to do. Since `Defect` and `Problem` are the same underlying concept, they must be reconciled. In addition, if the experts decide to include `Solution` in the consensus KB, John must provide his version of the rules that take the additional inference step, determining `Solution` from `Defect`. Those rules must then in turn be compared with Bruces's and any discrepancy resolved.

Matching Rules After reaching agreement on the ultimate goal for each KB, the system moves one step back in the inference chain in each, to see which attributes are used to determine the goal in each KB. It then uses the techniques just illustrated to get agreement on those attributes.

Once agreement on them is reached, the system begins to detect and remove incompleteness and inconsistency in the rules that link them. An example of detecting two apparently inconsistent rules arises in examining the rules that each expert used to determine the `Quality` of the model (Fig.5):

```
Now, let's try to get agreement on the rules determining QUALITY.
I have noticed the following discrepancy:

     BRUCE's KB                                JOHN's KB
IF F-TEST less-than CRITICAL-VALUE       IF F-TEST less-than CRITICAL-VALUE
THEN QUALITY of MODEL is POOR            THEN QUALITY of MODEL is FAIR

Since the vocabulary has already been discussed, I am pretty sure this
problem is not due to a failure to consolidate POOR and FAIR, or a
misunderstanding about the meaning of F-TEST or QUALITY.
Do you want to include both rules in the consensus KB? N
```

Fig. 5: Two inconsistent rules

CARTER knows six ways in which this sort of discrepancy---conflicting rules---can arise:

1. There is a misunderstanding about the vocabulary: `Poor` and `Fair` could be synonyms, hence the rules are actually identical. This is ruled out here because the experts have already agreed on the vocabulary (in a part of the interaction not shown).

2. There is not really a mismatch because both rules should be in both knowledge bases (each expert forgot one rule that the other remembered).[8]

3. One of the rules is incorrect and should be removed.

4. There is not sufficient precision in the vocabulary terms to differentiate between the two outcomes, e.g., perhaps we need more than a single threshold.

5. There is a chain of inference between `F-Test` and `Quality` that has not been made explicit in the current KBs and the discrepancy lies somewhere along that chain.

6. Both rules are over-generalized as stated: they are both missing an attribute whose value constitutes an important unstated assumption that the experts know, but forgot to make explicit.

After ruling out the first five explanations when asked by CARTER, the experts agree that the last possibility is correct, and that they omitted information about the `R-squared` statistic (Fig. 6).

```
Here is another possible solution:

   It is possible that you two are making different assumptions about
   some other attribute important for determining the conclusion of the
   rule, but that neither of you has included in your KB?  Take a
   moment to think about and discuss what other factors might be
   important in determining QUALITY.

Can you think of any other factors?  Y
What do you want to call this factor?  R-SQUARED
```

Fig. 6: Beginning to resolve the difference.

In response they elaborate their rules to include this, and discover that they had different underlying assumptions about the value of the `R-squared` statistic. The two revised rules turn out to be acceptable to both experts (Fig. 7).

[8] If `QUALITY` were known to be single-valued, this too could be ruled out, but this version of CARTER didn't have that information.

```
OK.  BRUCE, what value for R-SQUARED did you have in mind when
you wrote your rule?  NOT-SIGNIFICANT

JOHN, what value for R-SQUARED did you have in mind when you
wrote your rule?  SIGNIFICANT

If we included R-SQUARED in the two rules, they would look like this:

      BRUCE's KB                          JOHN's KB
IF F-TEST less-than CRITICAL-VALUE   IF F-TEST less-than CRITICAL-VALUE
  & R-SQUARED is NOT-SIGNIFICANT       & R-SQUARED is SIGNIFICANT
THEN QUALITY is of MODEL is POOR     THEN QUALITY of MODEL is FAIR

Are both of you willing to include both of these rules in the
consensus KB?  Y
```

Fig. 7: Completing the resolution.

CARTER then guides the experts in resolving remaining details about the new attribute R-squared, and continues to look for other places where the two knowledge bases differ. The techniques described above allowed CARTER to find roughly 65% of the known discrepancies in the two KBs and allowed it to guide the two experts to consensus on a subset of rules concerned with the problem of multicollinearity.

5. MATCHING REPRESENTATIONS: CIRCUMSTANTIAL EVIDENCE IS INEVITABLE

Rules provide a good way to express inferences, but are not specifically designed to capture and represent meaning. Hence CARTER must attempt to infer the overlap in meaning between two concepts by collecting circumstantial evidence from the knowledge bases themselves, e.g., comparing how these concepts are used in the rules and how they relate to other vocabulary terms. The difficulty here is in using circumstantial evidence to infer whether two concepts were intended to have the same or different meanings.

Why not then just have definitions for each concept in the knowledge base and compare the definitions? That is, build the original domain knowledge bases so that they contain not only rules built from attribute-object-value triples, but also have formal definitions for each concept (i.e., each attribute, object, and value), perhaps definitions of the sort created with predicate calculus, or other formal representation language (e.g., Brachman et al., 1978). Given formal definitions of the concepts, it would appear that we could compare those definitions directly and avoid the need for circumstantial evidence.

But the problem is unavoidable. If the domain knowledge bases have formal definitions for their terms, this immediately brings up the question of where those definitions came from. If they had

already been agreed upon by the experts, then we are dealing with a situation in which there is a consensus vocabulary already established. CARTER would have to deal only with rule differences, and a substantial part of the consensus task would already have been done. While this is a plausible situation, it is one in which most of the consensus problem has been solved before we get there.

If, on the other hand, each knowledge base has definitions for its concepts, but these were *created independently* (and have not yet been jointly agreed on), the same fundamental problem arises: given two concepts represented in independently developed KBs, the only grounds on which to determine whether they are likely to be the same is the circumstantial evidence arising from comparing their representations.

The key observation is that two experts working independently will construct ontologies (ways of looking at the world) that differ in some respects simply because they were constructed independently. Given two independently constructed ontologies, the only grounds on which to decide whether two concepts are the same is the circumstantial evidence available from comparing their representations.

A specific example from a common sense domain will help illustrate the point. Assume our first expert defines the concept of a `satisfied-parent` as "a parent with at least two children, all of whom are professionals." A KL-ONE style representation of this would look something like:

```
satisfied-parent      a-k-o      parent
          role        children   (value-restriction professional)
                                 (number-restriction >= 2)
```

A parent would be further defined as `a-k-o human`, etc.

The second expert, working independently, decides to define what we as onlookers happen know is the same concept, but phrases it slightly differently: a `happy-parent` is "a parent with at least two offspring, all of whom have professional degrees:"

```
happy-parent     a-k-o       parent
        role     offspring   (value-restriction professional-degreed)
                             (number-restriction >= 2)
```

Now suppose CARTER attempted to match up `satisfied-parent` in one KB with `happy-parent` in the other, in an effort to determine whether they actually meant the same thing (i.e., the experts would agree that they had the same concept in mind). The only form of evidence available to the machine is the circumstantial evidence arising from matches (and mismatches) in the representation structure.

For example, both concepts are `a-k-o parent`, lending some support to the notion that they are the same, but certainly not proving so. Note that even this evidence is in turn dependent on believing that `parent` is intended to capture the same concept in both knowledge bases (likely, but not a foregone conclusion). The only way to determine that in turn would be by matching the definition structures for `parent`, etc., applying the same matching process on up the a-k-o chain.[9]

In matching the rest of the definition of these two terms, the machine would discover that one of them has a role called `children`, while the other has a role called `offspring`. The process would once again have to subgoal on gathering evidence about whether those two were in turn the same.

Hence the fundamental task involved in matching two independently constructed representations is unavoidably one of gathering circumstantial evidence from the representation structures themselves.

5.1. The Utility of Formal Definitions

While inferring meaning from syntactic evidence is inevitable, there is some benefit in using a representation that provides a facility for definitions of terms. One obvious reason for this is that when we want to compare meanings, the task is easier if we have a representation designed to capture meaning, rather than one designed primarily to capture inference (i.e., rules)

A more subtle form of benefit arises because representation languages designed to express formal definitions often provide a carefully chosen set of representational primitives (e.g., `a-k-o, role, value-restriction, number-restriction`). To see how these can help, consider two different reasons why two experts might construct two different representations of the same concept. First of course, they may think about the world differently, perhaps creating different `a-k-o` hierarchies.

Second and somewhat more subtly, they may each have a slightly *different understanding* of what `a-k-o` is supposed to mean. That is, the consensus problem recurs here in a subtle way because the experts might not have a shared understanding of *the representation language itself*, and may write different representations of the same concept even when their "internal" ontologies match.

A carefully chosen set of representation primitives can raise the likelihood that the meanings of those primitives are themselves shared between the experts, thereby reducing the chance for error in the translation from mental construct to written representation structure. This also means that a system like CARTER can justifiably put more emphasis on the overlaps that it finds when working with those representations.

Hence using representations that provide formal definitions of meaning may improve the system's ability to match concepts. But the fundamental problem is inescapable: when working

[9]Note the similarity to tracing back along the inference chain when matching concepts in the rule-based system.

with two independently created knowledge bases, we inevitably have to compare meaning by matching representation structures and accumulating circumstantial evidence, no matter what representation language is used.

6. RELATED WORK

Several efforts share some similarity in general spirit with ours. The Delphi technique (Helmer and Rescher, 1959) for instance, uses debugging to achieve consensus among a group of experts on a specific issue. It is a three-step, iterative process involving, (i) submitting individual opinions and supporting reasoning to a skilled facilitator, (ii) preparation of a summary report by the facilitator, and (iii) forwarding of the report back to the experts. Here the facilitator plays a role in debugging by attempting to clarify the specific areas of disagreement among the experts. Although entirely manual, this approach, like CARTER, aids by help to set the agenda for the experts' discussion.

We have also drawn on the work of Brown and Burton (1978) and Kuper (1989) in conceiving of the problem in terms of detecting bugs in knowledge and associating a repair procedure with each bug.

Work in knowledge acquisition based on the repertory grid notion (Boose, 1986; Plaza et al., 1987; Gaines and Shaw, 1989) focuses on using knowledge of multiple experts, but offers only modest guidance for reaching consensus, providing little knowledge about either detection or repair.

Work by Reboh (1983) suggests using the explanatory facilities of rule-based systems to pinpoint areas of disagreement between two systems, but leaves it to the user to decide which point of view to accept. Work by Mittal and Dym (1985) explores the utility of interviewing multiple experts as a way of understanding a task better, but does not deal with combining their knowledge.

Some of the work on the CONSUL system (Mark, 1980) addressed the issue of finding good matches and discrepancies in concept definitions, and shared the intuition that the system might determine whether two concepts were closely related by comparing their representations. In this case the discrepancy problem typically arose when adding new concepts to the knowledge base, and the two participants in the negotiation were the existing knowledge base and the human expert who supplied the new concept.

A recent study (Klein, 1989) addresses the issue of resolving conflicting design specifications. Although their typology of conflicts is similar in some respects to our discrepancy catalog, their primary focus is on reconciling the designs themselves rather than design knowledge.

More distantly related is work on argumentation methods, aimed at assisting the process of discussion by helping people specify the logical structure of their positions. These ideas have recently been embodied in computer-based tools (e.g., Lowe, 1985; Smolensky et al., 1987; Stefik et al., 1987; Nunnamaker et al., 1988), but they are almost entirely process oriented: they assist experts in the process of deliberating and debating, but, importantly, do not suggest

resolutions to inconsistencies.

7. CONTRIBUTIONS

One primary contribution of this work is the catalog of 35 varieties of discrepancy, a store of detailed information we have codified for facilitating CKA. It represents a small but growing and relatively systematic expression of knowledge about how to detect and resolve disagreements in knowledge.

A second contribution arises from the surprisingly effective degree of bootstrapping displayed by CARTER. The system must make its best guess about the meaning of a term from the way it is used in a knowledge base, it can gather only circumstantial evidence of the sort reviewed above, and it must, paradoxically, gather that evidence from the very same knowledge bases it is attempting to modify to reach consensus. It it thus quite interesting how effective the system's heuristics are at guiding it. CARTER manages to make plausible judgments about which concepts match, so that even when it cannot be sure and has to ask the experts, the questions are for the most part sensible and well chosen.

A third contribution is the extensibility of our discrepancy catalog, illustrated during a session with the two statisticians that revealed the need for a new category of entry: differences in problem solving strategy. Where one expert looked for all possible problems, then suggested a solution, the other searched for and corrected one problem at a time. This new kind of discrepancy turns out to be detectable in terms of a difference in the topology of the two KBs (specifically, a difference in the number of goals); the detection/resolution organization of the catalog allowed us to add the appropriate new entries with minimal effort.

A final contribution is the potential of our work as a general approach to constructing systems that detect and resolve knowledge-level discrepancies. While our current system removes discrepancies in knowledge expressed in rules and attribute-object-value triples, we believe debugging and repair strategies can equally well be organized around other kinds of knowledge representations. The related work done using KL-ONE (Brachman et al., 1978) strongly suggests the plausibility of this approach. The fundamental process involves specifying the components of the representation, developing a taxonomy of how the representations can differ across these components, and prescribing possible resolutions for each of these discrepancies. We anticipate testing this on other forms of representation.

8. FUTURE WORK

One of the most important areas for future research is the question of discrepancy resolution strategy. While our strategy of starting at the outcome and working backward is useful, it is one of several possibilities. One problem is that it may be a bit too myopic to be effective in a large scale knowledge base. The system in effect immediately dives into the details and it needs a better sense of the larger picture. Our next task is thus to generate a number of strategies and evaluate them in terms of the efficiency and effectiveness with which they increase the degree of consensus, and the naturalness and coherence of the dialogues they produce.

Alternative strategies include working forward from inputs (because the two KBs are likely to start reasoning from the same basic information) and beginning at an intermediate point of agreement and expanding in both directions (emphasizing what the experts already agree on and building from there).

Another important area for future investiation is the problem of resolving expertise more explicitly at the knowledge level (Newell, 1982). A tool like CARTER compares expertise at the level of specific data structures in the knowledge bases (e.g. rules and attribute-object-value triples). Although CARTER can aid experts in detecting more abstract knowledge-level discrepancies (e.g. differences in problem-solving strategies), to do so it must "decompile" this information from the imlemented knowledge bases, which can be difficult. Recent work in knowledge acquisition (e.g. Musen, 1989) suggests that knowledge-base development can be significantly facilitated by providing tools that make problem-solving strategies explicit. Tools for CKA that operate on knowledge expressed in these terms may be particularly useful in the earlier stages of knowledge base construction, when the problem is still being defined. By contrast, tools such as CARTER may be most useful in later stages, when the experts must agree on specific terms and decision rules.

9. CONCLUSION

We have described a novel approach to and prototype system for facilitating consensus knowledge acquisition. The key contributions of this work include the development of a detailed store of knowledge for detecting and resolving discrepancies in rule-based systems, a general procedure for developing similar systems for other representations, and the empirical result that circumstantial evidence in the knowledge base itself can provide a useful degree of power in determining concept meaning. We expect the next advance in this area to come from implementing improved discrepancy resolution strategies. This work will serve as the starting point for understanding more generally how experts reach consensus and how we can best support them in their efforts to do so.

Acknowledgements

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124. Additional support for this research was provided by the International Financial Services Research Center of the MIT Sloan School of Management, Digital Equipment Corporation, McDonnell Douglas Space Systems, General Dynamics Corp, and the National Sciences and Engineering Research Council of Canada.

REFERENCES

Aczel, J., and Saaty, T. (1983). "Procedure for Synthesizing Ratio Judgments," *Journal of Mathematical Psychology*, **27** (1), pp. 93-102.

Boose, J. (1986). *Expertise Transfer for Expert Systems Design.* Amsterdam: Elsevier.

Brachman, R. et. al. (1978). ''KL-ONE Reference Manual,'' BBN Report No. 3848.

Brown, J., and Burton, R. (1978). "Diagnostic Models for Procedural Bugs in Basic Mathematical Skills," *Cognitive Science* **2** (2), pp. 155-192.

Gaglio, S. et al. (1985). "Multiperson Decision Aspects in the Construction of Expert Systems," *IEEE Transactions on Systems, Man, and Cybernetics* **15** (4), pp. 536-539.

Gaines, B., and Shaw, M. (1989). ,"Comparing the conceptual systems of experts," *Proceedings of IJCAI-89*, pp. 633-638.

Helmer, O., and Rescher, N. (1959). "On the Epistemology of the Inexact Sciences," *Management Science* **6** (1), pp. 25-52.

Jagannathan, V., and Elmaghraby, A. (1985). "MEDKAT: Multiple Expert Delphi-Based Knowledge Acquisition Tool" Engineering Mathematics and Computer Science Department Technical Report, Univ. of Louisville.

Klein, M., et al. (1989). "Towards a Theory of Conflict Resolution in Cooperative Design," *Proceedings of the 9th Workshop on Distributed Artificial Intelligence*, , pp. 329-349.

Kuper, R. (1989). "Dependency-Directed Localization of Software Bugs," MIT-AI-TR-1053, MIT AI Lab.

Lowe, D. (1985). "Co-operative Structuring of Information: The Representation of Reasoning and Debate," *International Journal of Man-Machine Studies* **23** (2), pp. 97-111.

Mark, W. (1980). "Rule-based inferences in large knowledge bases," *Proceedings of AAAI-80*, pp. 190-194.

Miner, F. (1984). "Group vs. Individual Decision Making: An Investigation of Performances Measures, Decision Strategies, and Process Losses/Gains," *Organizational Behavior and Human Decision Processes* **33** (1), pp. 112-125.

Mittal, S., and Dym, C.L. (1985). "Knowledge Acquisition from Multiple Experts," *The AI Magazine* **6** (2), pp. 32-36.

Musen, M. (1989). "Conceptual models of interactive knowledge acquisition tools," *Knowledge Acquisition* **1** (1), pp. 73-88.

Newell, A. (1982). "The Knowledge Level," *Artificial Intelligence* **18** (1), pp. 87-127.

Nunnamaker, J., et al. (1988). "Computer-aided Deliberation: Model Management and Group Decision Support," *Operations Research* **36**, (6), pp. 826-848.

Plaza, E. et al. (1987). "Consensus and Knowledge Acquisition," in Bouchon, B., and Yager, R. (eds.), *Uncertainty in Knowledge--Based Systems.* Berlin: Springer-Verlag, 1987. pp. 294--306.

Reboh R. (1983). "Extracting useful advice from conflicting expertise," *Proceedings of the 8th IJCAI*, pp. 145-150.

Smolensky, P. et al. (1987). "Constraint-Based Hypertext for Argumentation," Department of Computer Science and Linguistics, University of Colorado WP CU-CS-358-87.

Stefik M, et al. (1987). "Beyond the chalkboard," *Communications of the ACM* **30** (1), pp.32-47.

Trice A. (1990). "Facilitating Consensus Knowledge Acquisition," Unpublished Doctoral Dissertation, Massachusetts Institute of Technology, May 1990..

Toulmin, S. (1958). *The Uses of Argument.* Cambridge, England: Cambridge University Press.

Appendix: Listing of discrepancy catalog

Below is a listing of the 35 discrepancies in the discrepancy catalog, with a representative example drawn from the statistical domain and a possible resolution. (Note: Not all examples actually arose in the KBs used to test CARTER).

I. Vocabulary Discrepancies

Objects:

1. Same Object Concept, Different Spelling
   Example: `Model` vs. `Modell`
   Resolution: choose correct spelling
2. Same Object Concept, Different Name
   Example: `Model` vs. `Function`
   Resolution: pick a shared name
3. Object/Attribute Grouping Difference
   Example: `Regression` of `Dataset` is `Poor` (concept is an attribute)vs.
   `Quality` of `Regression` is `Poor` (concept is an object)
   Resolution: Change attribute to an object, or object to attribute
4. Same Object Representation, Different Level of Abstraction
   Example: `Regression` (with attributes `F-TEST, R-SQUARED`) vs.
   `Model` (with same attributes)
   Resolution: Model is a component of a Regression;use `Model` and omit extra level of representation
5. Extra Object
   Example: `IV-Correlation-Matrix` only found in one KB
   Resolution: Include in KB lacking concept, or delete

Attributes:

6. Same Attribute Concept, Different Spelling
   Example: `R.SQ` vs. `R-SQUARED`
   Resolution: choose correct spelling
7. Same Attribute Concept, Different Name
   Example: `F.Stat` vs. `F-Test`
   Resolution: pick a shared name

8. Attribute/Value Grouping Difference
   Example: `Heteroscedasticity` of `Model` is `Yes` (concept is an attribute) vs.
   `Problems` of `Model` is `Heteroscedasticity` (concept is a value)
   Resolution: Change value to an attribute, or attribute to a value
9. Same Attribute Representation, Different Level of Abstraction
   Example: `Heteroscedasticity.Solution` (with value `Log.Transform`) vs.
   `Next-Step` (with value `Log.Transform`)
   Resolution: `Heteroscedasiticity.Solution` is a more specific concept:; use `Next-Step` and omit extra level of representation
10. Extra Attribute
   Example: `Autocorrelation` only found in one KB
   Resolution: Include in KB lacking concept, or delete

Values:

11. Same Value Concept, Different Spelling
   Example: `Statistics` vs. `Stats`
   Resolution: choose one spelling
12. Same Value Concept, Different Name
   Example: `Exceeds-critical-level` vs. `Significant`
   Resolution: choose a shared name
13. Extra Value
   Example: `Fair` only found in one KB
   Resolution: Include in KB lacking concept, or delete

II. Topology Discrepancies

14. Extra Data Operation
   Example: `. . . → . . . DEFECT` vs.
   `. . . → . . . PROBLEM → SOLUTION`
   Resolution: Include extra attribute (and add extra rules that use it), or delete extra attribute
15. More Primitive Input
   Example: `F-TEST →  QUALITY` (`F-TEST` is an input to the system) vs.
   `QUALITY` (`QUALITY` is an input to the system)
   Resolution: Include extra attribute (and add extra rules that use it) or delete extra attribute
16. Finer Reasoning
   Example: `R-SQUARED → EXPLANATORY-POWER → QUALITY` vs.
   `R-SQUARED → QUALITY`
   Resolution: Add in extra reasoning step, or delete it; restructure rules as necessary
17. Different Decision Factors
   Example: `F-TEST, S-SQUARED → QUALITY` vs.
   `F-TEST → QUALITY`
   Resolution: Add in or remove discrepant attributes; restructure rules as necessary

III. Rule discrepancies

Rules with differing certainty factors:

Example:

```
IF SIGN of COEFFICIENT is SURPRISING vs.   IF SIGN of COEFFICIENT is SURPRISING
THEN PROBLEM of MODEL is                   THEN PROBLEM of MODEL is
     MULTICOLLINEARITY .7                       MULTICOLLINEARITY .5
```

18. Incorrect Rule Value(s)
   Resolution: choose a mutually agreed upon CF for rule (e.g. .6)

19. Insufficient Scale Granularity
    Resolution: allow the attribute SIGN to take on additional values (e.g. VERY-SURPRISING) and employ this new value in one of the rules.
20. Implicit Reasoning Steps
    Resolution: break up the one reasoning step into two, and compare the more explicit lines of reasoning
21. Implicit Assumption
    Resolution: find another factor important in determining PROBLEM (e.g. R-SQUARED), and add it into the premise of the rules.

Rules with the same premises, but differing conclusions:

Example:

```
IF SIGN of COEFFICIENT is SURPRISING vs.    IF SIGN of COEFFICIENT is SURPRISING
THEN PROBLEM of MODEL is                    THEN PROBLEM of MODEL is
      OUTLIERS                                    MULTICOLLINEARITY
```

22. Mutual Omission
    Resolution: both rules are correct: include both in consensus KB
23. Incorrect Rule Value(s)
    Resolution: choose mutually agreed on correct value (e.g. just MULTICOLlINEARITY)
24-26. Insufficient Scale Granularity, Implicit Reasoning Steps, Implicit Assumption
    Resolutions: analogous to #19, 20, and 21 respectively

Rules with the same conclusion, but different premises:

Example

```
IF SIGN of COEFFICIENT is INTUITIVE vs.     IF SIGN of COEFFICIENT is SURPRISING
THEN PROBLEM of MODEL is                    THEN PROBLEM of MODEL is
      MULTICOLLINEARITY                           MULTICOLLINEARITY
```

27. Mutual Omission
    Resolution: Analogous to #22
28. Incorrect Rule Value
    Resolution: Analogous to #23
29-31. Insufficient Scale Granularity, Implicit Reasoning Steps, Implicit Assumption
    Resolutions: Analogous to #19, #20, and #21 respectively

32. Residual (extra) Rule
    Example: any rule only found in one KB
    Resolution: delete, or include in consensus KB

III. Problem-solving strategy discrepancies

33. Different number of diagnoses per consultation
    Example: one KB has many more goal attributes than the other, suggesting that one solves more problems in a consultation than the other
    Resolution: Must align all goals of KBs: significant restucturing of KBs needed
34. Extra Strongly Indicative Information
    Example: One unmatched attribute (e.g. PRELIM-STEPS) is very close to the goal of a KB, suggesting that this information can sometimes be used by itself to make the correct diagnosis
    Resolution: Include extra attribute in consensus KB
35. Extra Solution
    Example: Analogous to #14

A Problem-Solving Model for Episodic Skeletal-Plan Refinement

Samson Tu, Yuval Shahar, John Dawes, James Winkles, Angel Puerta, Mark Musen
Medical Computer Science Group
Knowledge Systems Laboratory
Departments of Medicine and Computer Science
MSOB, Stanford University School of Medicine
Stanford, CA 94305-5479
(e-mail: tu@sumex-aim.stanford.edu)

ABSTRACT

PROTÉGÉ is a metalevel program that generates knowledge-acquisition tools that are based on the method of skeletal-plan refinement. In this paper, we propose a flexible and extensible architecture that allows the problem-solving method to be assembled from more basic methods. In this architecture, we emphasize (1) a uniform view of problem solving at different levels of granularity, (2) an explicit data model that allows construction of complex datatypes from predefined datatypes, and (3) the inclusion of domain-dependent control information within a domain-independent problem-solving method. We show how such a model of problem solving can drive the generation of knowledge-acquisition tools.

1 INTRODUCTION

Skeletal-plan refinement is a problem-solving method originally proposed for designing molecular-biology experiments (Friedland and Iwasaki, 1985). In the 1980s, we modified and extended the method for planning therapies for clinical trials (Tu et al., 1989). Because the method is used to instantiate skeletal plans at multiple time points, we call it the method of *episodic skeletal-plan refinement* (ESPR). The method was the basis for a series of experiments in building knowledge-acquisition systems. First, we built OPAL, a system for acquiring skeletal plans in the domain of cancer chemotherapy (Musen et al, 1987), then our laboratory developed PROTÉGÉ, a metalevel tool for generating knowledge-acquisition systems for clinical-trial management in alternative domains (Musen, 1989).

Like several other method-based knowledge-acquisition systems built at approximately the same time—ROGET for a type of heuristic classification (Bennett, 1985), MOLE for the cover-and-differentiate method (Eshelman, 1988), and SALT for the propose-and-revise method (Marcus and McDermott, 1989)—PROTÉGÉ assumes that there is a fixed method of problem solving that defines the roles that domain knowledge plays in the problem-solving process. PROTÉGÉ, unlike the other systems, divides the knowledge-acquisition process into two phases. First, a knowledge engineer maps the terms and relations describing the method's knowledge roles into concepts in the domain. In that process, she creates, in domain terms, a model of the knowledge required for solving the task using this method. Using this task model, PROTÉGÉ generates a knowledge-acquisition tool specialized for the task in the domain. Because the terms used in it are meaningful in the

domain, the knowledge-acquisition tool is intended for use by domain experts (who may be unfamiliar with the problem-solving method) to enter the detailed content knowledge. An inference engine called e-ONCOCIN interprets the resulting knowledge base to create an advice system (Figure 1).

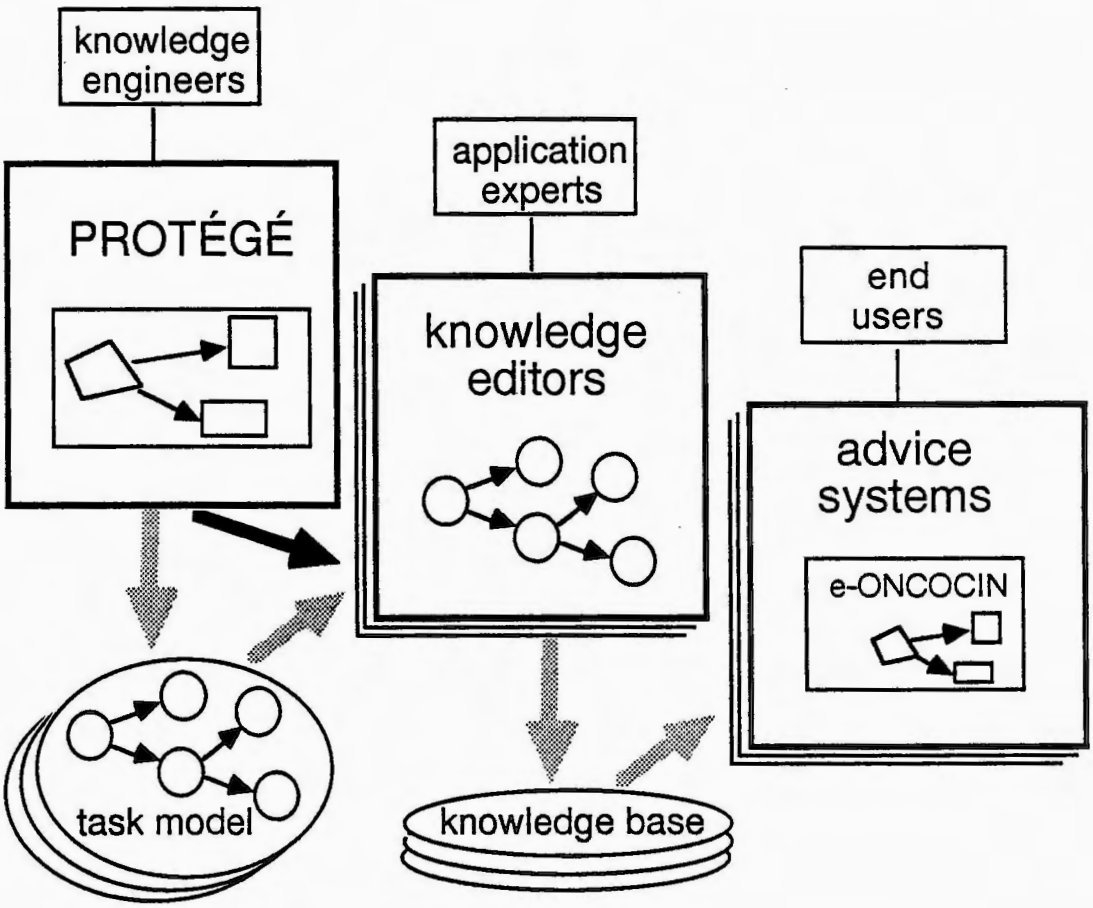


Figure 1: PROTÉGÉ system overview. PROTÉGÉ is a metalevel system that embodies the ESPR problem-solving method. Knowledge engineers use the terms and relations of the method to build domain-specific task models (shaded arrows) from which knowledge editors are generated (solid arrow). Application experts use the knowledge editors to develop knowledge bases that encode specific domain knowledge. End users consult with the advice systems to obtain recommendations. These advice systems apply the e-ONCOCIN inference engine—which implements the skeletal-plan–refinement method—to the knowledge bases to recommend appropriate planning actions.

As formulated in PROTÉGÉ, the ESPR method uses a collection of time-varying *planning entities* to represent the operators that can be applied, *input data* that are gathered from the environment, and *task actions* that can modify the instantiation of currently active planning entities in response to particular input data. In the domain of cancer treatment, for example, the planning entities might be chemotherapies and radiation therapies; the input data might be results of laboratory tests and physical examinations; and task actions might be rules for reducing the medication doses based on observations of drug toxicities. These

treatment options and rules for dose adjustments are often described in *protocols* that are followed by physicians. Thus, in treatment for a cancer, the domain-specific knowledge-acquisition tool generated by PROTÉGÉ is a tool for acquiring protocols that specify both different types of chemotherapies and radiation therapies and alternative ways of modifying the therapeutic actions in response to input data.

This methodology of using a fixed problem-solving method to define knowledge roles, although powerful, has a number of limitations. The method, being embodied in the code of the e-ONCOCIN inference engine, is not easily extensible.When the domain task to be solved by the method is described, there may be domain-dependent attributes that do not fit into any of the predefined roles of the method. For example, in cancer chemotherapy, the administered chemotherapies are counted in terms of cycles. Thus, an oncologist can speak of "second cycle of VAM chemotherapy." Yet the domain-independent ESPR method provides no guidance for defining such small-grained domain-dependent concepts. The knowledge engineer has to provide operational semantics for such domain-dependent terms by writing production rules in the inference engine's rule language. A similar problem arises because there is no easy way, except by writing symbol-level production rules, to add data-abstraction behavior to the ESPR method that is presupposed by PROTÉGÉ. The source of power of a methodology can be a limitation when the problem to be solved does not fit exactly into the framework of the methodology. The separation of control knowledge in the problem-solving method and content knowledge in the domain allows us to build knowledge-acquisition tools. Nevertheless, there are tasks for which some specific types of domain-dependent control knowledge are needed. In cancer treatment, one example of such domain-dependent control knowledge involves the sequencing of multiple chemotherapies over time. In PROTÉGÉ and in the knowledge editors generated by PROTÉGÉ, we showed that it is possible to integrate such information in our knowledge-acquisition systems. Yet this possibility was not extended systematically to other parts of the problem-solving method, such as the ordering of dose-reduction task actions, where knowledge engineers have had to order the underlying production rules manually to obtain the right behavior.

Another problem in PROTÉGÉ—although not one that is an intrinsic weakness of the methodology—is that PROTÉGÉ's data model does not permit the user to represent relationships between past input data and the planning entities that are modeled as temporal intervals. Thus, even though the representation language in the e-ONCOCIN problem solver allows the knowledge engineer to specify a temporal relationship such as "FIRST white-blood-cell count AFTER LAST chemotherapy," such a reference cannot be entered through the knowledge editor.

These considerations have led us to design a new PROTÉGÉ architecture, called PROTÉGÉ-II, that is configurable and extensible by the users both of PROTÉGÉ and of the knowledge editors created by PROTÉGÉ. It allows knowledge engineers to develop knowledge editors that are based on problem-solving methods other than ESPR, and to enter domain-dependent control knowledge in constrained ways. Furthermore, it has a consistent data model and a uniform view of problem solving at different levels of granularity.

We shall describe our approach to developing a model of problem-solving behavior that satisfies these objectives (Section 2). We propose that knowledge engineers may be able to configure a model of problem solving from a library of subcomponents that can serve as building blocks for custom-tailored problem-solving methods at different levels of granularity. We shall reformulate the ESPR method as a composite method assembled from other methods (Section 3). Finally, we shall show how the two-stage knowledge-acquisition process is applied in the PROTÉGÉ-II architecture (Section 4). In this paper, we

focus on the reformulation of the ESPR method in the PROTÉGÉ-II architecture. The design of PROTÉGÉ-II to implement the architecture, and the related human-computer interaction issues, are described elsewhere (Puerta et al., 1991).

2 The PROTÉGÉ-II ARCHITECTURE

We adopt a framework where problems to be solved are represented as *tasks*, and ways of solving them are represented as *methods*. A task is a specification of a problem to be solved in the world, such as diagnosing problems in a four-cylinder engine or designing a waveform generator. A *task class* is a family of tasks whose input and output specifications can be characterized in similar domain-independent terms. For diagnostic problem solving, the terms may be "symptoms" and "faults." For a design problem, the terms may be those describing the functional specifications of the required output and the available input components. In PROTÉGÉ-II, these input and output specifications, together with a task index, relations that the inputs and outputs must satisfy, and an optional predicate indicating when the task is completed, characterize the task class. No assumption is made about the procedure or the knowledge that is required for accomplishing the task.

Abstracting a task class from tasks applied in different domains is useful when we can define problem-solving methods that construct solutions for tasks in the task class. A problem-solving method is a partially predefined procedural specification for transforming input data objects into output data objects according to transformation rules that have meaning at the knowledge level. The input data objects include not only the case data on which the problem solver is working, but also domain knowledge that plays definite roles in the transformation rules. In the ESPR method that we shall describe (Section 4), the inputs include the case data, the skeletal plan that has been selected for the case, and the current time of the session. The output of the method is a set of planning-entity instances that represent real-world actions executable by the user of the advice system.

A method either is decomposable into a set of subtasks or is a basic method, which we call a *mechanism*. Each of the subtasks may have one or more methods for its solution (Figure 2). This recursive decomposition of methods into subtasks gives us a uniform view of the problem-solving process at different levels of granularity. Note that the level to which we decompose a method is a design decision regarding the appropriate level of abstraction. What is considered a mechanism in one context may well be considered a method in another. Our guideline is to define mechanisms that are sufficiently specific to solve particular tasks and yet sufficiently general to be reusable as parts of other problem-solving methods.

A method specification is analogous to a task specification. In addition to input and output declarations and semantic-constraint annotations, it has the following components:

1. A global data model that specifies the type of data and knowledge processed by the method
2. A collection of subtasks, and relations specifying how the subtasks' inputs are to be bound from the inputs of the method
3. A control structure for the subtasks, embedded in the code of an interpreter for the method, or specified explicitly in the task control language (for the purpose of the paper, the task-control language is simply a partial ordering among the subtasks)

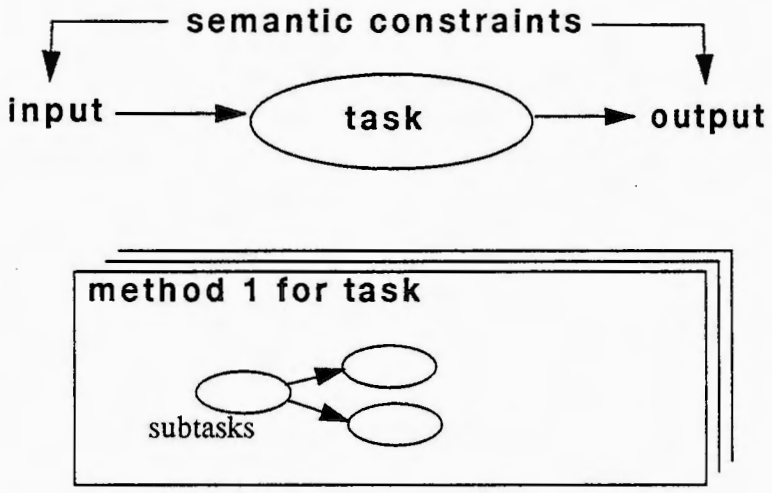


Figure 2. Task, method, and task decomposition. In the PROTÉGÉ-II framework, a task is characterized by its inputs, outputs, and semantic-constraint annotations. A task may be solved by multiple methods, which may decompose the task into subtasks, or the methods may solve the task directly.

A method is partially predefined in the sense that the input and output data objects on which the method operates must be specialized into those that have meaning to the domain expert, and the methods or mechanisms for solving the method's subtasks must be selected and ordered. As methods have different knowledge requirements, some methods may not be appropriate for configuring the problem solver in a particular domain. On the other hand, multiple methods may be appropriate for a single task. It is the knowledge engineer's responsibility to identify the appropriate methods for the tasks on which she is working. We call this process of data specialization and method selection and ordering *method configuration.*

PROTÉGÉ-II will have libraries of prebuilt task classes and methods. When appropriate, knowledge engineers will configure methods for tasks in particular domains. However, the domain task to be solved sometimes will not have an appropriate prebuilt problem-solving method. In this case, either there is nothing in the libraries that fits the task and the knowledge engineer needs to develop a new task description and its associated methods, or the knowledge engineer will *assemble* existing methods into composite methods.

To assemble composite methods from other methods, we must have a language for specifying terms and relations in a method. This method-specification language must have a *data model* for specifying the knowledge and data processed by the method, a *control model* for selecting and sequencing the execution of the subtasks or submethods in the problem-solving process, and a language for describing the semantics of the relationships among the inputs and outputs of the tasks and of the methods. The data model allows a knowledge engineer to declare, as input to the constructed task and method, datatypes that are composed from the datatypes associated with the existing tasks and methods. The data model also specifies the kinds of query and updates that can be performed on the data. The control model allows her to partition the required control information, some of which she

specifies as she assembles the problem-solving method, while leaving other parts for the domain expert to enter.

An analogy to a programming language illustrates the need for these three aspects of the problem-solver definition. A programming language provides a set of basic datatypes (such as strings and structures) from which application data structures can be declared. Similarly, each problem-solving method must presuppose some general model of the data it processes. That data model provides the base datatype language with which knowledge engineers can define other concepts in the domain. A programming language has control structures such as *for* loops and *if–then* statements. At the level of the problem-solving process, we must specify the ordering of the invocation of the subtasks and their methods, either statically or dynamically. Finally, a programming language has formal and informal specifications of the semantics of its operations. We do not expect to be able to develop a language for formally specifying the behaviors of the problem solvers. However, with each task and method, we shall associate descriptive annotations on the properties of its inputs and outputs. With each method, we shall associate similar annotations on its knowledge requirements and expected problem-solving behavior.

3 COMPONENTS OF THE ESPR METHOD

We shall illustrate the architecture for PROTÉGÉ-II by describing in detail how we have reformulated the ESPR method in the new framework. While the ESPR method was built into the original PROTÉGÉ, it can be assembled from those methods and mechanisms in the built-in method library of PROTÉGÉ-II. We shall describe first the global data model that is assumed throughout, then the partitioning of the control information into the component that is prebuilt into the methods, the component that is configured by the knowledge engineer, and the component that requires domain expertise. Finally, we shall describe the subtasks and the methods from which the ESPR method can be assembled.

3.1 Global Data Model

The nature of the cases that an expert system solves depends on the task and on the domain. Those cases, and the solutions to those cases, must be represented by a collection of data structures. Consequently, a problem-solving method such as episodic skeletal-plan refinement must presuppose some general model of the data it processes. That data model provides the base datatype language with which knowledge engineers can define other concepts in the domain. The global data model is a property of a class of domains and is not unique to any particular problem-solving method. We assume that each method presupposes a global data model whose primitives can be specialized—but not augmented—by the PROTÉGÉ-II user.

Time Model: A function of the global data model is to define the appropriate temporal properties of data. In the domain of medical therapy planning for which PROTÉGÉ was originally designed, patient data can be time-invariant (such as the patient's social security number), time-stamped (such as laboratory-test results), or interval-based (such as a treatment intervention that takes place over time). In the model of time for skeletal-plan refinement, we have adopted the time point as the primitive temporal element. We assume that time points have granularities expressed as a calendar/clock time unit. The use of time points with granularities implies that we have adopted a model of discrete time points at each level of time granularity. We assume that the granularity units are chosen such that a time point specified at a finer level can always be converted to a coarse level. For example, "12-May-89" can be converted to "May-89." In this model, intervals are defined as a pair

of time points, and the semantics of the intervals will be defined in terms of the corresponding time-point pairs.

There are three special time points: PAST, FUTURE, and NOW. For all time points t, t is after PAST and before FUTURE. NOW represents the "current time" (or today's date, if date is the granularity of time) associated with a particular patient case in an interaction with the problem solver.

We define precedence relations over time points, granularities, and durations. These precedence relations must take into account granularities of the time stamp. For example, we define two types of equalities between time points with different granularities. In one equality relation ($\approx$), the two time points are compared at their common granularity (thus, "1990 $\approx$ June 1990"); the other equality relation (same_time) requires that time points be compared at their finer granularity (thus, "1990 same_time June 1990" is not true). For time intervals, we have defined corresponding relations based on the relations on their start and stop points.

In a time model for skeletal-plan refinement, assertions are interpreted over time points. Thus, it is possible to ask whether something is true at a particular time. If an assertion is associated with an interval, then that assertion holds for all time points in the interval.

Entities: In the data model that we are using for our reformulation of ESPR, the primitive data elements are entities with attributes. The entities are class objects that may have instances; the attributes of the class objects may be either class attributes whose values are common to all instances or instance attributes whose values must be determined for each instance.

We want to be able to define basic relations among the class objects, such as the *kind of* relation (e.g., a prescription for AZT drug is a *kind of* medication, which is a *kind of* planning entity), or the *part-of* relation, (e.g., white-blood-cell count is *part-of* hematology lab results).

Classes representing concepts in the domain have a temporal specification of static, time-stamped, or interval. Instances of the two latter classes have corresponding instance attributes of either *time-stamp*, or *interval*, the latter of which is a pair of time points. Thus, the planning entity *medication* has instances that have the attributes *start time* and *stop time*, which denote the beginning and end of drug-administration episodes.

Expressions and Data Manipulations: In our new architecture, we take a pragmatic approach in specifying the global data model associated with a given problem solver. The global data model consists of a grammar for datatype declaration, data access, and data manipulation, with an interpreter that implements the semantics of that grammar. The grammar provides the primitive datatypes, basic terminals and nonterminals, and rules for constructing legal expressions for data access, data manipulation, and more complex datatypes. The grammar thus constitutes a modeling language in terms of which the concepts and the primitive operations in the domain can be defined.

For example, the grammar allows expressions such as "FIRST value of white-blood-cell count DURING LAST medication INTERVAL," where the capitalized key words have meaning within the grammar for the particular data model, and the lower-case symbols are terms that either must be predefined in the method specification (e.g., *value*), or must be defined by the knowledge engineer for the domain (e.g., *white-blood-cell count* and *medication*). The grammar and its interpreter allow the semantics of the expression to be defined in terms of the data model, rather than in terms of ad hoc symbol-level constructs.

For example, one way in which the interval associated with a planning entity could be represented would be as an *n*-tuple in a relational table that stores the start and stop times and the granularities associated with those time measurements; the data model, however, shields the knowledge engineer and the domain expert from such symbol-level concerns.

The grammar for the global data model also defines the legal data manipulations. Entities that are associated with time intervals may have instances that are manipulated with temporal operators that start a new interval (creating an instance of the entity with a certain start time), that stop an interval (setting the stop time to a particular time point, thus clipping the interval), that delete an interval (removing an instance of the entity from the system's active memory), and that alter the attribute of an interval-associated entity (setting some attribute of the instance to a particular value).

3.2 Control Model

The control model of this architecture relies on a model of agenda-based execution (Firby, 1989). The run-time model of the problem solver consists of an agenda to which problem-solving tasks can be posted. The tasks can have priorities and they can be linked by a partial ordering. An agenda interpreter cyclically determines the priorities of the tasks subject to the ordering constraints. The methods or mechanisms associated with the selected task in turn are selected for execution and linked to the invocation task. Method- or mechanism-specific interpreters execute the methods or mechanisms, and possibly post more tasks to the agenda.

As the tasks and methods are executed, they incrementally write their output to a global working memory. It is not necessary for the task or method to be completed before other methods are triggered by changes in the working memory. Thus, the input and output specifications of the tasks and methods are data-flow relationships.

This agenda-based execution model allows great flexibility in specifying control information. The control information can be implicit procedural control hidden in the code of the method or mechanism interpreter, or it can be expressed explicitly in a control language that orders the subtasks in a method's subtask decomposition. Alternatively, the triggering conditions in methods or mechanisms may cause data-driven execution. In specifying the control structure in skeletal-plan refinement, we shall use all three types of control schemes.

We have developed languages for specifying both domain-dependent and domain-independent control knowledge. One example of domain-dependent control specification is the transition diagram shown in Figure 3. It specifies the chemotherapy sequence for a particular cancer clinical-trial protocol. In PROTÉGÉ-II, this control information is the input knowledge to a planning-entity decomposition mechanism that is invoked when the system attempts to instantiate the appropriate chemotherapy at a point in time. Thus, the role of the domain-specific control knowledge is defined by the mechanisms that specifically use that kind of knowledge. Our current domain-independent control language allows the specification of a partial ordering among a set of subtasks.

3.3 Subtask and Method Specification

We see the ESPR method as being decomposed into three subtasks: proposing plan actions based on a high-level plan, identifying problems, and modifying the plan action in light of the identified problems. We shall use these three subtasks and their methods to illustrate how we specify tasks, methods for solving them, and the domain and control knowledge

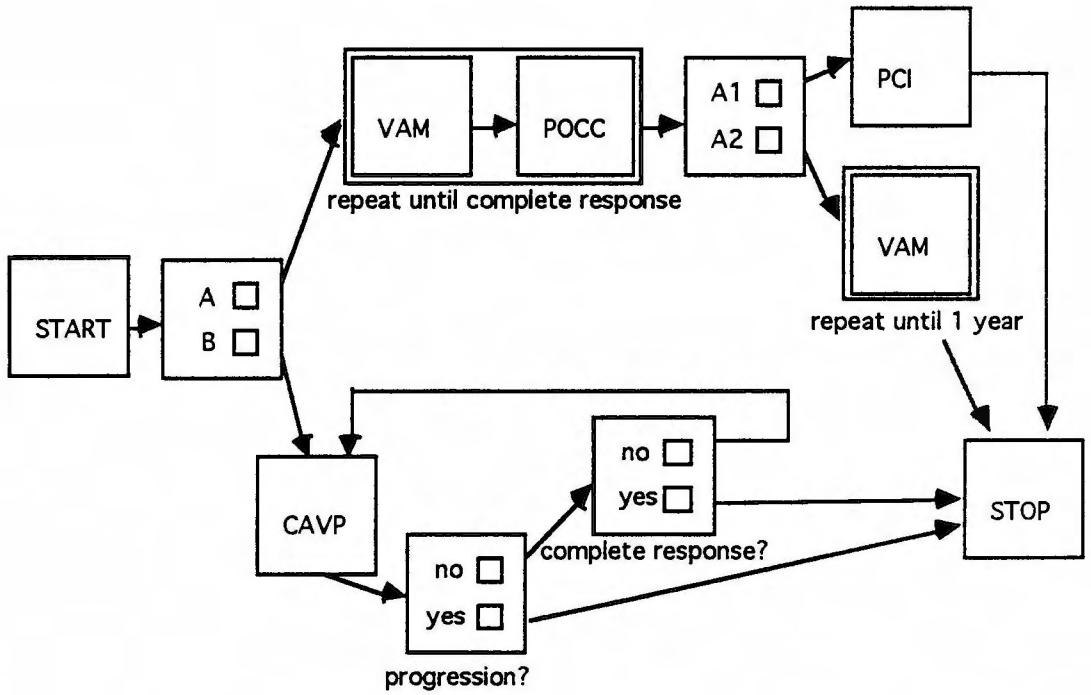


Figure 3. Example of a schema for a cancer-therapy protocol. This figure shows a simplified version of the sequences of steps in a protocol for lung-cancer therapy. Patients are randomly assigned to two possible treatment sequences, called arms A and B. In arm A, the chemotherapies VAM and POCC are given; in arm B, CAVP is given. Those patients who respond to treatment in arm A are divided into two subgroups. Patients in one subgroup receive prophylactic cranial radiation (PCI), whereas those in the second subgroup do not. One year after patients are enrolled in the protocol, their disease status is reevaluated.

that must be acquired to generate both the knowledge-acquisition system and the problem solver. We shall assume that these three subtasks and the methods for solving them have already been entered into our task and method libraries. Furthermore, we shall assume that all the small-grained subtasks, methods, and mechanisms for configuring these methods are in the built-in libraries. In the Section 4, we shall show how the ESPR method for the protocol-based execution planning task can be assembled from these subtasks and methods.

All three subtasks have the case data and a time point as inputs. The plan-proposal task, in addition, takes as an input a planning-entity class. The task then produces a set of planning-entity instances that have been instantiated from the input planning-entity class and from other planning-entity classes that make the given planning entity executable. The problem-identification task takes case data and a set of problems to identify as input, and generates a set of intervals representing the problems that are detected in the case data. The plan-modification task uses problems that have been identified and a set of planning-entity instances as input, and generates a new set of planning-entity instances.

In protocol-based execution planning, we use three methods—*instantiate-and-decompose*, *generalize-from-data*, and *situation-directed-revision*—to solve the plan-proposal, problem-identification, and plan-modification subtasks, respectively. These three methods use knowledge of the planning entities, the problem-abstraction mechanisms, and a set of situation-based mappings from case data to plan-modification subtasks. These three types of knowledge together constitute our model of a protocol.

The *instantiate-and-decompose* method for proposing an execution plan has as its inputs case data, a time point, and a top-level planning entity. Its subtasks are to instantiate the given top-level planning entity and to use one of the decomposition mechanisms to find those component parts of the planning entity that must be instantiated recursively at the given time point.

The *generalize-from-data* method for identifying problems uses case data and a control structure for identifying the problems from the given case data. In PROTÉGÉ's library of methods, we shall have a set of problem-detection and abstraction mechanisms.[1]

The *situation-directed-revision* method uses case data and the problems identified in the problem-identification subtask to modify the planning entities proposed in the *instantiate-and-decompose* phase. Associated with the method is a parametrized plan-modification subtask, where the parameter is one or more of the domain-dependent problems (such as renal toxicity or opportunistic infection) that are specified by the knowledge engineer or the domain expert. The control structure of this method binds problems identified in the problem-identification phase to the parametrized plan-modification subtask. For each possible combination of problems, the domain expert needs to specify the appropriate plan-modification procedure, either using the primitive data-manipulation operations in the data model or specifying another plan-proposal subtask.

4 KNOWLEDGE ACQUISTION FOR THE ESPR METHOD

The use of a global data model, a control model, and libraries of domain-independent tasks and methods as the basis for expert-system development has implications for building a knowledge-acquisition system.

The knowledge-acquisition system must have facilities that allow the entry of the data query and manipulation expression allowed in the grammar of the global data model. This facility is needed everywhere that the specification of an expression is required and, thus, needs to be implemented as an editor that can be embedded in other editors.

Moreover, the knowledge engineer will be defining the domain concepts used in the methods by *subclassing* the input and output classes of tasks and methods or by *composing* existing classes into composite classes. Subclassing an entity class involves specifying values in some class attributes and defining additional domain-specific attributes. The PROTÉGÉ-II system uses the input and output class declarations of the prebuilt tasks and methods, and generates forms that allow the knowledge engineer to define specializations of those classes. These domain-specific subclasses defined by the knowledge engineer in turn determine the forms in the knowledge editor that PROTÉGÉ-II creates for the domain experts.

---

[1] See (Shahar, 1991) for definitions of the temporal-abstraction mechanisms and the requirements for instantiating these mechanisms.

Composing a class means declaring a new class and specifying that the attributes of that class take existing class objects (or their subclasses) as values. In the example to be discussed, we shall declare a *protocol* as a composite class whose attributes are planning entities, problems, and plan-modification actions. The problem-solving methods that act on composite classes, such as the *protocol*, access the attributes of the composite classes through predefined class access functions provided by the data model.

The control model in PROTÉGÉ-II includes languages for specifying both domain-independent ordering of subtasks and domain-dependent flow of control. Both languages must have associated editors through which the control information can be entered. As the domain-dependent control-flow editor will be used by application specialists, the possible datatypes used by the editor must be couched in domain terms (e.g., sequencing of *medications* to administer). Thus, the control-flow editor must be instantiated for a particular domain.

Finally, the knowledge-acquisition system must have the facility to allow a knowledge engineer to associate methods from the method library of PROTÉGÉ-II with the tasks and subtasks selected for the problem-solver. The knowledge acquisition system must index the methods and tasks to help the knowledge engineer select appropriate methods for the tasks (Puerta et al., 1991).

Using these facilities provided by PROTÉGÉ-II, the knowledge engineer first assembles and configures the problem-solving method for the task (Section 4.1). The input and output specifications of these configured methods define the knowledge roles for which domain-specific datatypes must be defined (Section 4.2). Using the configured methods and mechanisms and the domain-specific knowledge roles defined by the knowledge engineer, PROTÉGÉ-II generates a knowledge editor that can be used by application specialist to enter the necessary domain knowledge (Section 4.3).

4.1 Assembling the ESPR Method

To assemble the ESPR method, the knowledge engineer must make a number of design decisions. As we have seen, there are a number of data-flow relationships among the subtasks. These data-flow relationships only partially constrain the possible control flow. For example, it is possible that, before the propose-plan subtask has completed execution, the problem-identification subtask can be invoked with some problems to check. The knowledge engineer can fix the order of the invocation of the subtasks by placing a partial ordering on the subtasks. In Figure 4, we force the plan-modification subtask to execute only after the initial plan-proposal subtask has completed, but we place no such constraints between the plan-proposal and problem-identification subtasks.[2]

The output specifications of the configured method can be derived easily from the outputs of the method's constituent subtasks. The knowledge engineer should select a subset from the union of the outputs of the constituent subtasks. In the ESPR method that we are configuring, we may choose the problems the system has identified, as well as the planning-entity instances, as the outputs of the method.

[2] A more general architecture will have metalevel control methods (strategies) that dynamically determine the order of the execution of the subtasks.

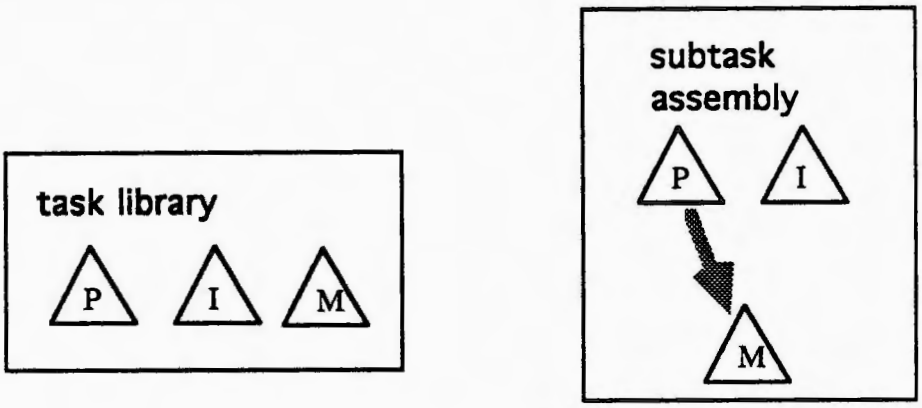


Figure 4: How a composite method is assembled from subtasks in the library. (a) Built-in tasks in PROTÉGÉ. (b) Assembled method. M is constrained to be invoked only after P has completed execution.

The inputs to the top-level constructed method are more difficult to define. It is not sufficient to take the union of the inputs of the constituent subtasks. Recall that the input data objects of a method include not only the case data, but also the knowledge used in solving the problem. For PROTÉGÉ-II to generate a knowledge editor for a clinical-trial management task, it must have a coherent model of the protocol in the domain. Thus, the inputs to the ESPR method that we are constructing cannot be specified completely until the knowledge requirements for methods that solve the subtasks are known. It is from these knowledge requirements that we construct a model of the protocol and any other domain knowledge required for solving the protocol-based execution-planning task.

To develop a model of the protocol, we must abstract the knowledge used in all the methods and mechanisms that may be used to solve the subtasks in the task decomposition of the assembled ESPR method. The knowledge engineer must use the class-composition capability of our data model to declare a protocol as a class object that contains (1) a set of planning entities and their associated decomposition mechanisms, (2) a set of problems and their associated problem-identification mechanisms, and (3) a set of plan-modification mechanisms.

4.2 Defining the Domain Task Model

Once the overall problem-solving method has been selected or assembled, knowledge acquisition is driven by the requirements of operationalizing the method in a particular domain. The inputs of the methods must be specialized to terms meaningful in the domain. For illustration purposes, we shall use clinical-trials of drugs for HIV-positive patients as the application domain.

Given the input requirements of the ESPR method assembled in Section 4.1, PROTÉGÉ-II shall ask the knowledge engineer to define a set of case-data objects representing concepts in the domain (such as white-blood-cell counts or bilirubin), a *part-of* hierarchy of planning-entity classes, a set of possible problem abstractions, and a set of revision mechanisms that map problems and data to plan-modification actions. For each subclass that the PROTÉGÉ-II developer defines, she should define the class and instance attributes

associated with that subclass. For each attribute, the system needs to know the legal values as well as the time when the values are determined (knowledge-editor time or runtime). We shall briefly illustrate some of the planning entities and problem abstractions that a knowledge engineer may specify for the HIV domain.

Figure 5 shows one planning-entity part-of hierarchy for the HIV domain. This specification of planning entities indicates that protocols in this domain have three levels of abstractions for therapeutic actions: (1) protocol administration as a whole, (2) regimens for alternative types of treatment, and (3) medications making up a regimen. This structure shall determine the types of domain knowledge that the PROTÉGÉ-II-generated knowledge editor will request from the domain expert.

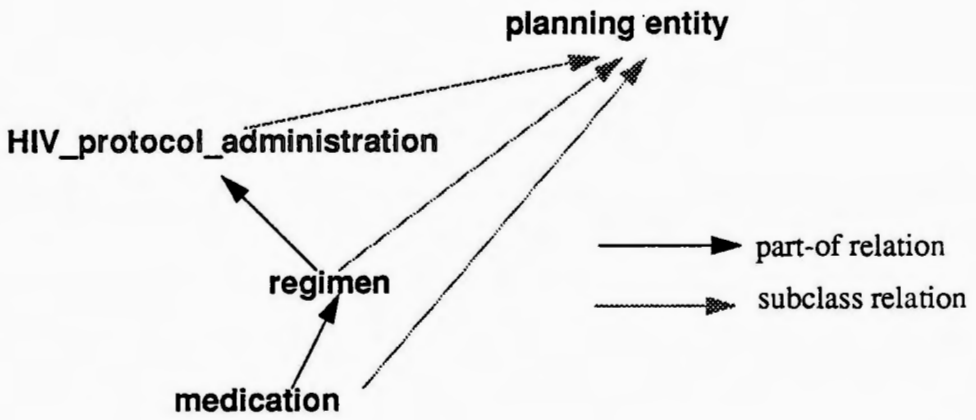


Figure 5: PROTÉGÉ-level specializations of the *planning-entity* class for the HIV domain. *HIV_protocol_administration, regimen,* and *medication* are declared to be subclasses of *planning-entity* class. *Regimen* contains *medication* while it is part of *HIV_protocol_administration* as class object.

For each class in the planning-entity composition hierarchy, the PROTÉGÉ-II knowledge engineer indicates what subtasks for finding instance-attribute values should be set up in the instantiation process. The semantics of these attributes will be defined by the mechanisms that the knowledge engineer associates with these subtasks. We have identified a small number of mechanisms for specifying relationships among attributes of planning entities. One such mechanism allows specification of definitional relationships among attributes of entities. For example, the current cumulative dose of the drug can be defined as the sum of a previous cumulative dose and the product of the current dose and frequency. Other relationships among attributes will be defined by the domain specialists using the mechanisms made available from the method library.

Figure 6 shows part of a problem hierarchy that may be defined for the HIV domain.[3] The problem list defined by the PROTÉGÉ developer forms the menu of problem classes that

[3] The problems shown in Figure 6 are abstractions about patient *states*. We have also developed mechanisms for making *gradient* and *rate* abstractions that characterize direction and rate of change of parameter data (Shahar et al. 1991).

the knowledge-editor user may use to define protocol-specific problems for which therapy actions may be altered.

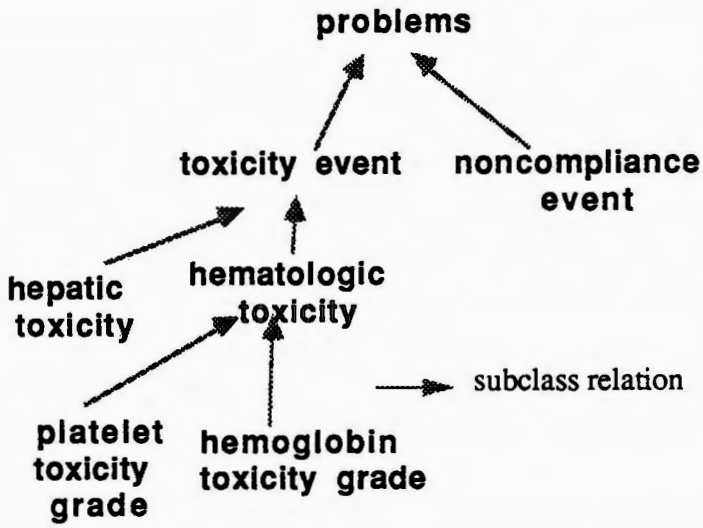


Figure 6: PROTÉGÉ-level specializations of the *problems* class for the HIV domain. The knowledge engineer specifies these potential problems that may arise during the administration of a clinical trial and provides default methods for instantiating these problem classes.

4.3 Knowledge Acquisition Using Knowledge Editors Generated by PROTÉGÉ-II

Given planning-entity and problem class hierarchies such as those shown in the previous section, PROTÉGÉ-II generates a knowledge editor customized for the domain. Instead of using generic terms like *planning entity*, the knowledge editor uses domain-specific terms such as *medication*.

For each domain-specific planning-entity class, the user of the knowledge editor defines subclasses that are used in that protocol. The knowledge editor has forms through which the attribute values of these planning entities can be specified. For planning entities that have part-of children, the user is required to specify the decomposition method by selecting from the library of mechanisms the appropriate decomposition mechanism, and to specify the necessary data for that mechanism.

Figure 7 shows how the planning entities of a protocol (protocol CCTG-522) can be defined as subclasses of the planning entities defined at the PROTÉGÉ level. A decomposition mechanism associated with a planning entity may instantiate some of the planning-entity at lower level, or a plan-modification mechanism may cause a planning entity to be instantiated. Each of the mechanisms has its own editor. For example, if the domain expert wants to specify a sequence of treatment actions similar to the one in Figure 3, the knowledge editor brings up the graphical editor for specifying such a procedure (Puerta et al. 1991).

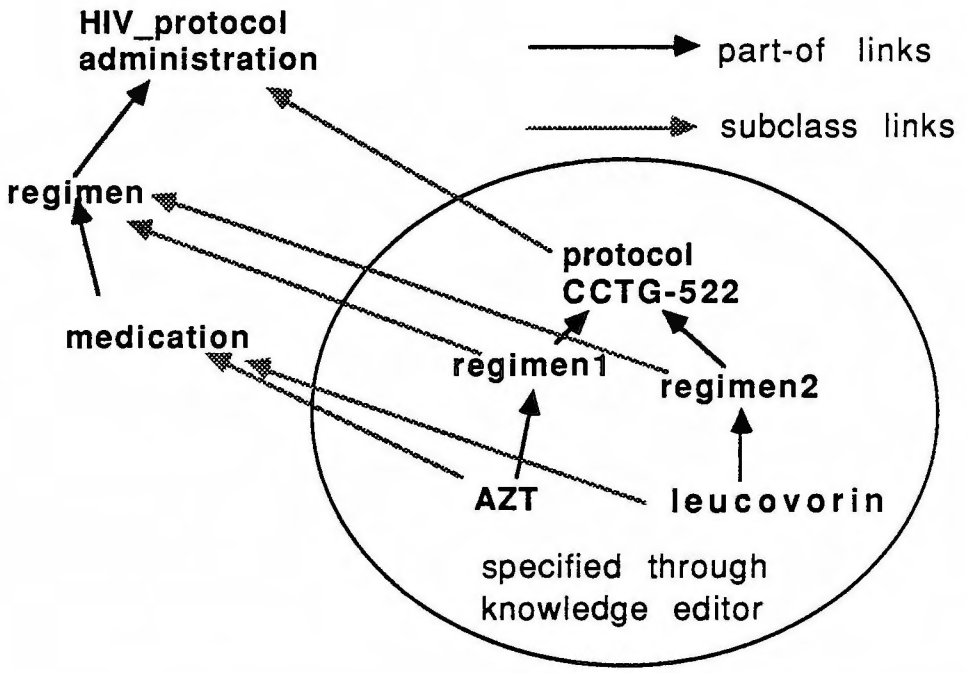


Figure 7: Specialization of HIV planning entities for a protocol. The user specifies that protocol CCTG-522 has two drug regimens. In the first regimen, patients receive the drug AZT, whereas in the second regimen, the drug leucovorin is given. The knowledge editor uses the domain-specific planning-entity classes specified in PROTÉGÉ-II to present customized interfaces for acquiring knowledge about particular protocols, regimens, and medications.

The knowledge-editor user selects from the problem class hierarchy to specify the problems that are relevant to a particular protocol (Figure 8). The user should be able to define protocol-specific concepts (e.g., a *dose-reducing-toxicity* in CCTG-522). There shall be a set of problem-abstraction mechanisms with which the domain expert can specify how a problem is to be derived from input data or lower-level abstractions (Shahar et al. 1991). The knowledge editor will check for completeness by making sure that an abstraction mechanism has been instantiated for every problem mentioned in a protocol.

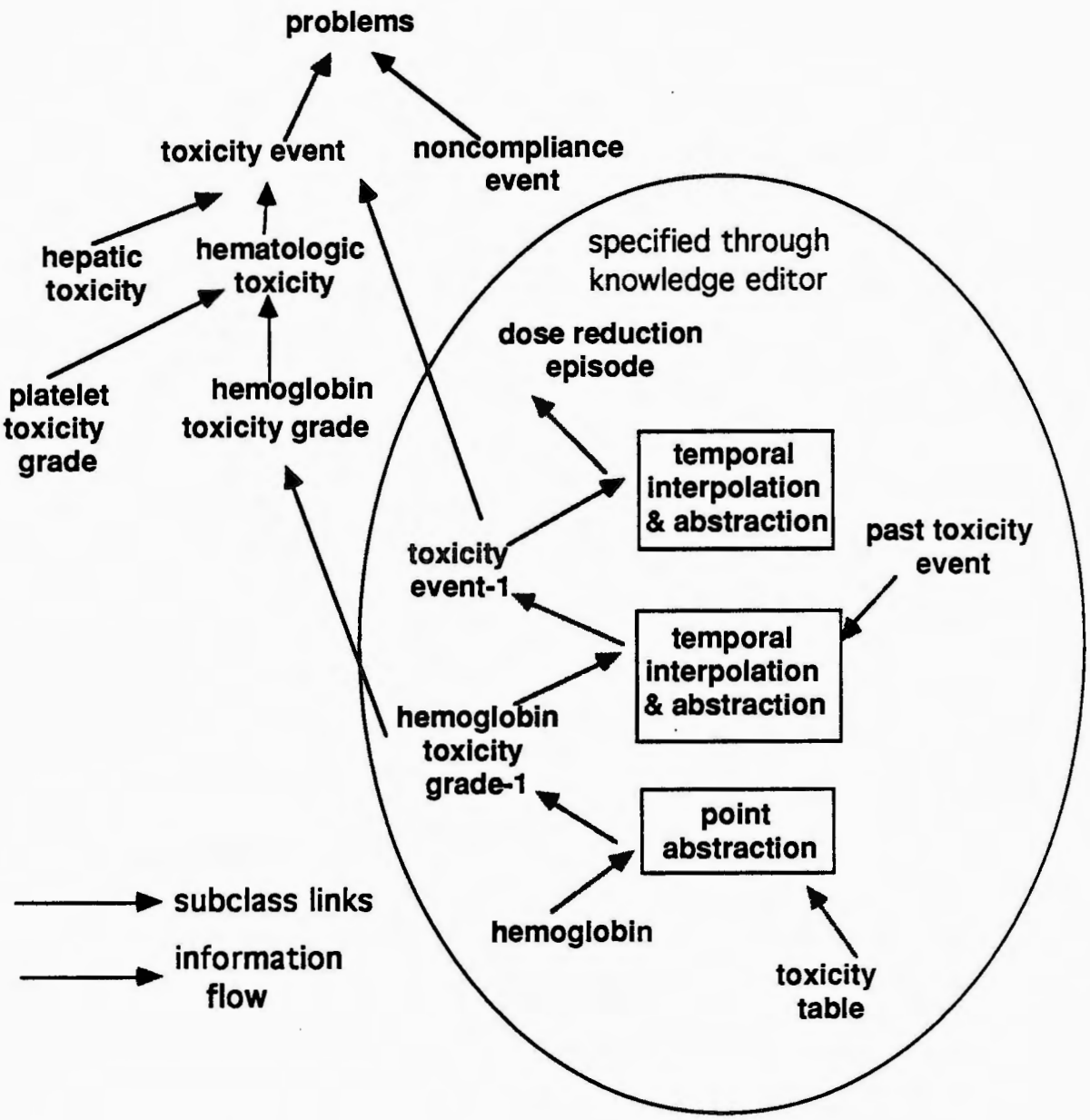


Figure 8: Use of abstraction mechanisms to define problems. The three abstraction mechanisms in the oval are knowledge-editor–level specifications of how the *dose-reduction episode*, *toxicity event-1*, and *hemoglobin toxicity grade-1* are to be determined from either data or lower-level abstractions. The problems to be determined are driven by the *problems* hierarchy that the knowledge engineer entered in PROTÉGÉ-II.

DISCUSSION

The first PROTÉGÉ system built in our laboratory assumed that the application task for which it was designed could be viewed in terms of the episodic skeletal-plan–refinement model. The tool provided a language for expressing domain concepts in terms that a special-purpose inference engine (e-ONCOCIN) could use. This assumption of a fixed method and the consequent fixed knowledge roles allowed a knowledge engineer working with PROTÉGÉ to generate graphical knowledge editors that were tailored to the application area. These editors could be used by domain specialists to enter the detailed content knowledge required by the problem solver.

This assumption of a fixed problem-solving method makes PROTÉGÉ a brittle system in two respects. First, the system by definition cannot be used for tasks and methods for which it is not designed. Even for a given task, there may be several alternative solution methods, each of which is better in some situations than in others. Second, a monolithic method cannot anticipate all possible variations in the ways that knowledge is applied in a domain. The method is applicable only in domains where the knowledge requirements fit the method exactly down to the most detailed level of abstraction. For those aspects of the knowledge that depend on domain-dependent considerations (for example, a procedure for calculating the current dose of a drug), the skeletal-planning model in PROTÉGÉ defines no explicit knowledge roles that users simply can fill in.

In our current work, we deal with the problem of brittleness in three ways. First, we generalize the PROTÉGÉ framework so that instead of using one fixed problem-solving method our system has libraries of tasks and methods for solving them. Configuring or assembling the problem-solving method becomes the first step in the knowledge engineer's interaction with PROTÉGÉ-II. This capability to tailor problem-solvers for alternative tasks extends the range of the tasks for which PROTÉGÉ-II can be used, and it allows the knowledge engineer to associate multiple methods with the same task. Second, the uniform view of problem-solving as using methods to accomplish tasks allows us to use the same type of knowledge-level analysis at multiple levels of granularity. For a small-grained task such as making an abstraction from data, we can define correspondingly small-grained domain-independent data-abstraction mechanisms. As a method is decomposed into subtasks that have their own solution methods, selecting a top-level method needs not fix the methods for the subtasks. This capability to define problem-solving methods at multiple levels of granularity allows great flexibility in customizing the problem-solving behavior for a particular task. Finally, we emphasize the use of a *data model* and a *control model* that provide general-purpose languages for configuring and assembling problem-solving methods. These are tools that a knowledge engineer uses for customizing the knowledge editors and the problem-solvers generated by PROTÉGÉ-II.

We allow for the specification of domain-dependent control knowledge by defining languages (such as the flowchart language used in Figure 3) for expressing such knowledge and specialized mechanisms that interpret them. These mechanisms are invoked within specific methods (e.g., the flowchart in Figure 3 is interpreted by a decomposition mechanism that is invoked only in the *instantiate-and-decompose* method). Thus, we allow the specification of domain-dependent control knowledge, but only in very constrained contexts. Such constrained use of domain-dependent control knowledge gives us the necessary flexibility while retaining the advantages of role-limiting methods.

We recognize that developing a knowledge-based system is a complex task requiring both the modeling expertise of a knowledge engineer and the domain expertise that only an application specialist can provide. Our solution to this dilemma, as was the case in the first version of PROTÉGÉ, is to divide the knowledge-acquisition process into two phases. In

the first phase, the knowledge engineer uses the modeling tools of PROTÉGÉ-II, such as the libraries of tasks and methods and data-model and control-model languages, to configure a problem-solving method for the task and the domain. Based on the knowledge requirements of the configured method, PROTÉGÉ-II generates a domain-specific knowledge editor that the domain specialist can use to enter the necessary domain knowledge.

Our group's project to seek ways of extending PROTÉGÉ's method-oriented architecture parallels work by a number of other researchers to develop ways in which problem-solving methods can be adapted more closely to the needs of particular domain tasks. For example, researchers at the Free University in Brussels have developed a componential framework (Steele, 1990) and implemented a prototype testbed (Vanwelkenhuysen, 1990) that bear similarities toto the PROTÉGÉ-II architecture. The componential framework requires the system developer to analyze a task in terms of input and output, available domain knowledge, case model, and pragmatic constraints, such as incompleteness of data, that may be present in the domain. A problem-solving method in the componential framework may have domain models that decompose a task into subtasks or it may solve a task directly. The implementation uses the a frame system that allows them to represent each component of the framework as an object that can be dynamically tailored to the problem at hand.

PROTÉGÉ-II differs from the componential framework more in degree than in kind. In the componential framework, domain knowledge is part of the task characterization, and the method contains the procedural control structure. In PROTÉGÉ-II, on the other hand, both the knowledge and the control structure are associated with the problem-solving method. While the componential framework is consistent with a two-phased approach to knowledge-acquisition such as PROTÉGÉ methodology, Steels (1990) sees the development of knowledge-based systems as primarily the responsibility of the knowledge engineer. Our emphasis on the division of labor between knowledge engineers and domain experts leads us toward developing a set of data and control models that knowledge engineers can use to configure and assemble methods and toward exploring techniques for generating domain-specific knowledge-acquisition systems from such configured methods.

Faced with brittleness problems in the Generic Task architecture (Chandrasekaran, 1986), investigators at Ohio State University are moving to an approach in which generic tasks are implemented in SOAR (Johnson et al., 1990). Each generic task is represented within a SOAR problem space (Laird et al., 1986). As such, the inputs to the generic task are modeled as the initial state of the problem space; the desired output of the generic task is modeled as the goal state; and the problem-solving method is modeled using operators available within the problem space. The Ohio State group hopes that SOAR's universal weak method will provide a common foundation for all generic tasks and that SOAR's universal subgoaling behavior will facilitate the integration of multiple generic tasks that each may contribute to problem solving at different levels of granularity. A significant advantage of the SOAR approach is that the weak methods in Soar provide a default problem solver when specific domain knowledge is not available. It is not clear, however, how the SOAR representation itself can make explicit the roles in which knowledge is used in problem solving to facilitate the generation of knowledge-acquisition tools like those created by PROTÉGÉ.

McDermott's group at Digital Equipment Corporation shares our view that problem-solving methods can be composed from more primitive building blocks called mechanisms (Klinker et al., 1991) and that these mechanisms can define distinct roles for the knowledge that knowledge-acquisition tools can supply. The group at Digital is developing a metatool called Spark, which will help a developer select mechanisms from a library and

will combine them to construct problem-solving methods accommodating the requirements of particular application tasks. The output of Spark is a knowledge-acquisition tool (called Burn) that will allow users to enter specific domain knowledge.

Despite strong similarity, PROTÉGÉ and Spark are being designed for different communities of users. Our research group assumes that knowledge engineers will be the principal users of PROTÉGÉ and that these engineers will want to configure new problem-solving methods explicitly. The developers of Spark, however, anticipate that the primary users of their tool will be nonprogrammers and, thus, are building Spark to select and modify configurations of mechanisms programmatically based on features of the application task (Klinker et al., 1991). Although Spark itself does not address the problem of how to guide knowledge engineers in their assembly of different mechanism configurations, providing such guidance is a major concern for the new version of PROTÉGÉ. Our group is experimenting with a number of visual languages that may help knowledge engineers to construct complex datatypes in the method configuration process and to view complex relationships among mechanisms (Puerta et al., 1991).

It is encouraging that several research groups are converging on similar methodologies for developing knowledge-based systems. In all these projects, considerable attention is given to identifying what problem-solving abstractions are appropriate and useful and how these small-grained abstractions might be combined into useful problem-solving methods. Although there are important differences in the manner in which the various architectures allow users to assemble and instantiate problem-solving methods, a primary challenge faced by all groups involves describing, at the knowledge level, appropriate tasks and methods. This common framework raises the possibility that these descriptions of problem-solving behavior can be shared across the research groups.

This paper reports on work of an on-going project. As we implement the problem-solving abstractions for the method of episodic skeletal-plan refinement and test similar ideas in other application areas, we will refine our languages for describing data, domain knowledge, and control information. We believe that the framework described in this paper will give us a basis for developing customized knowledge-acquisition tools and flexible and extensible problem solvers.

ACKNOWLEDGMENTS

This work has been supported in part by grant LM05157 from the National Library of Medicine, by grant HS06330 from the Agency for Health Care Policy and Research, by a grant from the Institute for Biological and Clinical Investigation at Stanford University, and by a gift from Digital Equipment Corporation. Computer support was provided in part by grant RR05353 from the Biomedical Research Support Grant Program of the National Institutes of Health and by the SUMEX-AIM resource, supported by grant LM05208 from the National Library of Medicine.

John Egar and Josef Schreiner have contributed greatly to our new design for PROTÉGÉ and have provided valuable comments on previous drafts of the paper.

REFERENCES

Chandrasekaran, B. (1986). Generic tasks for knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert* **1**(3):23–30.

Chandrasekaran, B. (1990). Design problem solving: A task analysis. *AI Magazine* **11**(4):59–71.

Firby, R.J. (1989). *Adaptive execution in complex dynamic worlds*. Ph.D. Dissertation, Department of Computer Science, Yale University, New Haven, CT.

Friedland, P.E., Iwasaki, Y. (1985). The concept and implementation of skeletal plans. *Journal of Automated Reasoning* **1**:161–208.

Johnson, T.R., Smith, J.W., Chandrasekaran, B. (1990). Task-specific architectures for flexible systems. Technical Report. Department of Computer Science, Ohio State University, Columbus, OH.

Klinker, G., Bhola, C., Dallemagne, G., Marques, D., McDermott, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition* **3**:117–135.

Laird, J., Rosenbloom, P., Newell, A. (1986). *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*. Boston: Kluwer Academic.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for Expert Systems*, S. Marcus, (ed.), pp. 225–256. Boston: Kluwer Academic.

Musen, M.A., Fagan, L.M., Combs, D.M., Shortliffe, E.H. (1987). Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man–Machine Studies* **26**:105–121.

Musen, M.A. (1989). *Automated Generation of Model-Based Knowledge-Acquisition Tools*. London: Pitman.

Puerta, A., Egar, J., Tu, S., Musen, M. (1991). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. In *Proceedings of the Sixth Knowledge-Acquisition for Knowledge-Based Systems Workshop*, ed. by Boose, J. H. and Gaines, B. R., Alberta, Canada, in press.

Shahar, Y., Tu, S.W., Musen, M.A. (1991). Knowledge acquisition for temporal abstraction mechanisms. In *Proceedings of the Sixth Knowledge-Acquisition for Knowledge-Based Systems Workshop*, ed. by Boose, J. H. and Gaines, B. R., Alberta, Canada, in press.

Steels, L. (1990). Components of expertise. *AI Magazine* **11**(2):30–49.

Tu, S.W., Kahn, M.G., Musen, M.A., Ferguson, J. C., Shortliffe, E.H., Fagan, L.M. (1989). Episodic monitoring of time-oriented data for heuristic skeletal-plan refinement. *Communications of the ACM* **32**:1439–1438.

Vanwelkenhuysen, J., Rademakers, P. (1990). Mapping a knowledge level analysis onto a computational framework. *Proceedings of the Tenth European Conference on Artificial Intelligence*, pp. 661–664. Stockholm, Sweden.

ACQUIRING FIRST-ORDER KNOWLEDGE ABOUT AIR TRAFFIC CONTROL

Christel Vrain* & **
** LIFO, rue de Chartres,
Université Orléans,
45067 Orléans cedex, France
email : cv@univ-orleans.fr

Yves Kodratoff*
* CNRS & Université Paris Sud, LRI,
Bldg 490, 91405 Orsay, France
email : yk@lri.lri.fr

Abstract
This paper presents an application of knowledge intensive generalization to knowledge acquisition in the domain of air traffic control. We insist on the large translation problem to be solved to go from the expert's language (a classical questionnaire language) up to a Horn clause description. Once the learning has been taking place, the acquired knowledge must be translated back into a language directly understandable by the expert. Solving these translation problems is very important in order to insure expert validation of the knowledge acquired automatically .

1 - INTRODUCTION

An air traffic controller performs many different tasks that can be simulated by different models. The problem of avoiding the crash of aircraft is an essential one and much work deals with it. Most of it concerns the development of automatic pilot aid (Speyer, 1988; Onken, 1988) or automatic controller tools (Hunt and Kloster, 1987; Phillips, 1987; Bleistein and al., 1987; Avizienis and Ball, 1987; Levi and al., 1987; Chaib-Draa and al., 1988); only few (Planchon and Berrada, 1988) have tried to model the controller task by means of an ES and as far as we know, the results were not completely satisfactory.
The goal of the project PERSPICACE[1] (Cannat and Vrain, 1988) is the application of ML techniques to learn ATC rules, in the case where expert rules cannot be obtained by means of direct communication with the expert. As it seems easier for an expert to give examples rather than rules of his behavior, we planned to make use of the achievements of the sub-field of ML (Michalski, 1983; Kodratoff and al. 1984; Kodratoff and Ganascia, 1986; Kodratoff and Tecuci, 1988) that deals with the problem of learning a recognition function of a concept by generalization of examples of this concept, and avoiding it covering negative examples. In this paper, we intend to show the problems that have arisen, mostly due to the necessity to use first order logics rather than an attribute-value representation (Vrain and Kodratoff, 1991) and to transform the information given by the expert into first order representations. They have been partly solved and we explain the solutions that we have adopted.
This paper uses a technique of generalization that can be described as **knowledge intensive generalization to first order.** In order to make clear what is meant by this, we will define what is knowledge intensive learning and describe the symbolic generalization tool used in this application.

2- KNOWLEDGE ACQUISITION AND LEARNING

2.1 - Knowledge intensive similarity based learning

Inductive generalization is the process which "reverses" the (deductive) generalization

---

[1]PERSPICACE : (from the French "Prototype d'Expert en Résolution de Situations. Problème Intéressant le Contrôle Aérien et Conçu à partir d'Exemples") contract C.E.N.A. / L.R.I., october 1986 - october 1988

inference rule, $[\forall x\ p(x)] \Rightarrow p(A)$, where A is a constant. Instead of deducing the instances from the general concept p(x), we induce p(x) from some of its instances $p(A_1)$, $p(A_2)$,..., $p(A_i)$. In general, one is given a set of instances, $p^1(A_1)$, $p^1(A_2)$,..., $p^j(A_i)$, one does not even know if they are related to the same concept.

- The problem of clustering the instances into groups illustrating one concept in each cluster has been called conceptual clustering (Stepp and Michalski, 1983; Kodratoff and Tecuci, 1988). In this paper, we do not cluster the examples of conflict resolution provided by the expert. It is part of the questioning to ask the expert himself to cluster the cases that "sound quite similar" to him. In other words, in the result presented in this paper, conceptual clustering is performed by hand by the human expert. Notice however, that the acquisition of this kind of knowledge is far from usual because other knowledge acquisition systems would be unable to take advantage of this information, supposing the expert is willing to provide it. Our experience shows that our experts were quite willing (actually, they found it entertaining) to provide such clustering information for which they have to use only their intuition without constraints.

- Once an automatic or manual clustering has been performed, the problem of describing the instances contained in the same cluster by their common properties is exactly the one of finding a common generalization for a given set of examples. There can be knowledge poor or knowledge intensive generalizations. Knowledge poor generalization, also called empirical learning, uses the examples in the form they are given (which actually does contain a large amount of information, but this information is implicit), proceeds by collecting the features that are shared by all the positive examples of the concept (Lebowitz, 1990). It happens also that negative examples are used, in that case the constraint is that no negative examples shares the features of the generalization of the examples. On the other hand, a knowledge intensive generalization method explicitly uses knowledge about the domain in order to refine the recognition function to learn. It takes into account explicit knowledge about the domain, and embodies it into the examples, thus expending the description of the examples, so as to take into account this explicit knowledge to fill up "holes" in the description of the examples.

Let us consider for instance the concept of *good driver*. and let us suppose that the following descriptions are given to us. "John is a man, he is forty years old, he is married and he has never had a car accident, and he is a good driver", "Mary is a good driver, she is a woman, she is fifty years old, she is married and she has never had a car accident", "Peter is twenty years old, he is a man and is bachelor, he had an accident, and he is a bad driver." The three events can be summed up by the following table.

	sex	age	married	accident	driver
John	male	40	yes	no	good
Mary	female	50	yes	no	good
Peter	male	20	no	yes	bad

Table 1

A knowledge poor generalization technique will collect the features that are common to the examples. Therefore, it would learn here that a good driver is married, and had no car accident. This method does not use explicit **background knowledge** (called **BK** throughout this paper), but it uses a large amount of implicit BK in order to transform the given sentences into the table above. On the other hand, a knowledge intensive generalization method explicitly uses knowledge about the domain in order to refine the recognition function to learn. There are many possible "generalization rules" (Michalski, 1983) that can be used to improve a generalization. In general, we shall say that this knowledge can be expressed as sets of theorems relative to the domain. Suppose that we know that a person who is between thirty and sixty years old is middle-aged, that is to say that we have theorems such as $[30 \leq age \leq 60 \Rightarrow$ middle-aged], $[16 \leq age \leq 29 \Rightarrow$ young]. Then, using this extra-knowledge, we can learn a

more precise recognition function of a good driver, that is, a good driver is married, had no traffic accidents, and is middle-aged.

2.2 - A presentation of OGUST

The system OGUST is a generalization tool which learns a recognition function of a concept from a set of examples. It is based on the principle of structural matching (Kodratoff and Ganascia, 1986) and it is knowledge intensive.

The representation language of the examples and of the generalization is restricted to the predicat logic. Functions of arity greater or equal to 1 are not allowed. For instance, let us suppose that a car insurance company wants to learn the concept of a "costly driver", prone to be expensive for the insurance company and let us work on the following two examples.

E1: John and Mary are married, they have their driving licence, John has a car, and Mary uses it only on holidays.

E2: Peter has a son Bob. Bob has just obtained his driving licence, sometimes he borrows his father's car.

There are many ways of translating these two sentences into predicate logic, depending on the one hand on the pieces of implicit information that we want to make explicit, and, on the other hand, on the choice of the predicate symbols and of the constants. We shall not discuss this problem now, since we show in section 3.2.4. how the choices have been made for the application to ATC. Following these choices, E1 and E2 are translated in the following way, where RE1 and RE2 denotes the representation of E1 and E2.

RE1 = married (JOHN, MARY) & driving_licence (JOHN) & driving_licence (MARY) & owns (JOHN, C1) & car (C1) & uses_on_holiday (MARY, C1)

RE2 = son (BOB, PETER) & new_driver (BOB) & owns (PETER, C2) & car (C2) & borrows (BOB, C2, PETER)

We notice here that we have represented only the knowledge explicitly expressed in the sentences. We could also represent that JOHN, PETER and BOB are men whereas MARY is a woman since these pieces of information are implicit in the sentences.

Let us suppose that BK is given by the following set of theorems.

R1: $\forall x$	man (x)	$\Rightarrow$	person (x)
R2: $\forall x$	woman (x)	$\Rightarrow$	person (x)
R3: $\forall x \forall y$	married (x, y)	$\Rightarrow$	family_relation (x, y)
R4: $\forall x \forall y$	son (x, y)	$\Rightarrow$	family_relation (x, y)
R5: $\forall x \forall y$	family_relation (x, y)	$\Rightarrow$	family_relation (y, x)
R6: $\forall x$	new_driver (x)	$\Rightarrow$	driving_licence (x)
R7: $\forall x \forall y$	uses_on_holidays (x, y) & car (y)	$\Rightarrow$	driver (x, y)
R8: $\forall x \forall y$	owns (x, y) & car (y)	$\Rightarrow$	driving_licence (x)
R9: $\forall x \forall y \forall z$	borrows (x, y, z)	$\Rightarrow$	driver (x, y)
R10: $\forall x \forall y$	uses_on_holidays (x, y) & car (y)	$\Rightarrow$	unskilled_driver (x)
R11: $\forall x$	new_driver(x)	$\Rightarrow$	unskilled_driver (x)
R12: etc.			

The generalization algorithm can be decomposed into two main steps, a deductive one followed by an inductive one.

- In the first step, the examples are put into structural matching (Kodratoff, 1983; Kodratoff and Ganascia 1986), that is to say, their representations are transformed using the background knowledge (BK) until they match or until no more transformations can be applied to improve the matching.

In order to understand this step, we can imagine that examples are collection of objects, each object being characterized by its own properties and its relations with other objects. For instance, the example E1 is composed of three objects JOHN, MARY and C1. JOHN is characterized by the property that he has a driving licence and by some relations with MARY and with C1. To apply structural matching to the examples, we have first to find among the objects those that are in some sense the most similar, i.e., the most promising for the future generalization. We have to make explicit the common properties between these objects.
In other words, structural matching can be decomposed into two steps

- the choice of an object in each example and the replacement of all their occurrences by a variable of generalization, this choice is based upon a function, sim(O1,O2), that computes the similarity between two objects,
- for those properties that do not match, the improvement of their matching as much as possible, using the given BK

These two steps iterate in a cycle which is repeated until there are no more objects to choose in the examples.

- The second step generalizes the two representations we have obtained by structural matching.

In order to exemplify these notions, consider again our concept of "costly driver", and examples E1 and E2.

2.2.1 - Structural matching (deductive step)
In E1, there are three constants, JOHN, MARY and C1. In E2 there are also three constants, BOB, PETER and C2. To determine which objects are the most similar, OGUST uses a heuristic comparing the explicit properties of the objects. In our example, our heuristic will have to compare JOHN and PETER, JOHN and BOB, JOHN and C2, MARY and PETER, MARY and BOB, etc. In order to decrease the number of comparisons to be performed, the constants of the examples can be typed. For instance, here, we could say that it is uninteresting to compare a person with a car, which will lower the number of comparisons from eight to four. To type the constants is another way of introducing knowledge in the process of generalization, it constrains the space of possible generalizations by discarding "uninteresting" over-generalizations.

If we do suppose that the constants are typed, we have only to compare JOHN and PETER, JOHN and BOB, MARY and PETER, MARY and JOHN. We shall not detail now the heuristics of choice, but simply give an intuitive feeling of the way they have been built. Intuitively, we can say that JOHN and PETER are the persons who seem the most similar, since they both own a car. Thus, we replace all the occurrences of JOHN and PETER by a same variable of generalization, called here vg1. In other words, we can say that JOHN and PETER will be generalized in the inductive step by the variable vg1. We get

RE1 = ***married*** ***(vg1***, MARY) & ***driving_licence*** (***vg1***) & driving_licence (MARY) & owns (vg1, C1) & car (C1) & uses_on_holiday (MARY, C1) & [(= vg1 JOHN)]

RE2 = ***son*** (BOB, ***vg1***) & new_driver (BOB) & owns (vg1, C2) & car (C2) & ***borrows*** (BOB, C2, ***vg1***) & [(= vg1 PETER)]

In each example, vg1 satisfies the underlined property of owning an object, these occurrences of vg1 are common to both examples. Since they belong to both examples, these occurrences of vg1 in the atom "owns (vg1, C1)" of E1, and in the atom "owns (vg1, C2)" of E2, are said to be not discriminating. On the contrary, in E1, vg1 has a driving licence and is married whereas in E2, vg1 has a son and someone borrows something from him/her. In that case, since vg1 occurs in atoms that do not belong to both examples, they are called discriminating

occurrences of the variable vg1. They are written in italic-bold in RE1 and RE2. The work of OGUST is to try cancelling these differences by using the available BK. For instance, we can prove in E2 that vg1 has a driving licence, by means of theorem R8. We cannot prove that in E2, *vg1* is married but we can prove that in E1 and in E2, *vg1* satisfies a family relation. Therefore, the discriminating occurrence stays as such, but we introduce new terms, stemming from BK, that are consequences of the information contained in the raw examples, and that decreases the discrimination between E1 and E2. We give here no details about the algorithm that perform this task, it is carefully described in Vrain (1990).

We get :

RE1 = *married (vg1*, MARY) & driving_licence (vg1) & driving_licence (MARY) & owns (vg1, C1) & car (C1) & uses_on_holiday (MARY, C1) & family_relation (vg1, MARY) & [(= vg1 JOHN)]

RE2 = *son* (BOB, *vg1)* & new_driver (BOB) & owns (vg1, C2) & car (C2) &*borrows* (BOB, C2, *vg1)* & driving_licence (vg1) & family_relation (vg1, BOB) & [(= vg1 PETER)]

The same process is iterated. We shall not detail it. We should then choose BOB and MARY, replace their occurrence by a new variable of generalization *vg2* and treat the discriminating occurrences of vg2 using BK. At least, we should replace all the occurrences of the last constants C1 and C2 by a variable vg3 and treat its discriminating ones. Finally, we get :

RE1 = married (vg1, vg2) & driving_licence (vg1) & driving_licence (vg2) & owns (vg1, vg3) & car (vg3) & *uses_on_holiday* (vg2,*vg3*) & family_relation (vg1, vg2) & driver (vg2, vg3) & unskilled_driver (vg2) & [(= vg1 JOHN) (= vg2 MARY) (= vg3 C1)]

RE2 = son (vg2, vg1) & new_driver (vg2) & owns (vg1, vg3) & car (vg3) &*borrows* (vg2, *vg3*, vg1) & driving_licence (vg1) & family_relation (vg1, vg2) & driver (vg2, vg3) & unskilled_driver (vg2) & [(= vg1 PETER) (= vg2 BOB) (= vg3 C2)]

We have no more constants in the examples and we have tried to remove as much as possible the discriminating properties, given our BK. Thus, the structural matching step is completed.

2.2.2 - Inductive step

We generalize the two examples in a straightforward way, by keeping their common properties. This yields the recognition function G.

G = owns (vg1, vg3) & car (vg3) & driving_licence (vg1) & family_relation (vg1, vg2) & driver (vg2, vg3) & unskilled_driver (vg2)

2.2.3 - Conclusion

The main advantages of OGUST are :

- its representation language. Predicate logic is richer than an attribute-value representation, since it represents efficiently several objects and the relations between them,
- its systematic use of BK in order to find a recognition function of the concept as specific as possible,
- its ability to generate explanations of its results.

Its drawbacks are :

- its exponential complexity. In order to find a generalization of the examples, it studies all the alternate choices of objects to match,
- its problems to deal with negative examples. Heuristics have been elaborated to choose the constants to match according to the negative examples, but they do not enable to treat all of them, and the recognition function we learn may not be consistent,

- its inability to deal with numeric values.

2.3 - Knowledge acquisition and Learning

Similarity-based learning can be applied to knowledge acquisition according to the scheme, presented in figure 1.

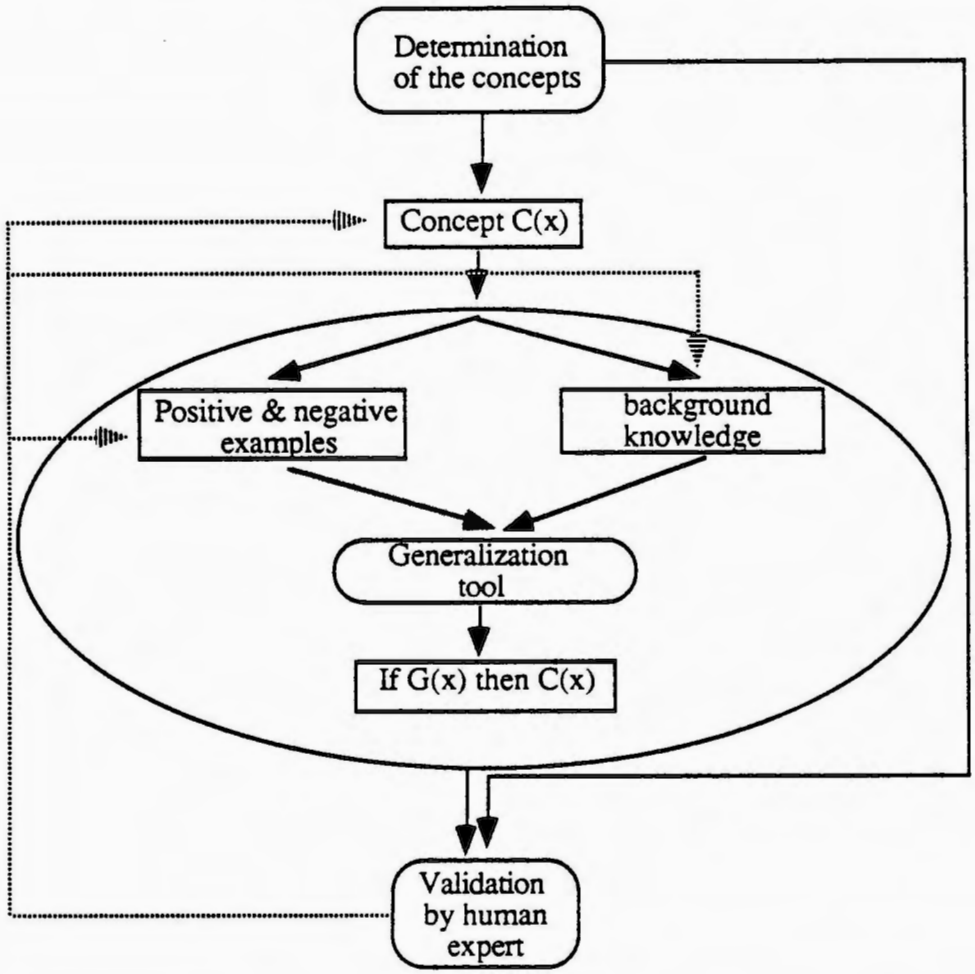


Figure 1. *The cycle of acquisition/validation of knowledge.*

The dialogue with the expert will then be divided into four main steps. At the beginning of the process, we need three steps: to acquire BK, to determine the concepts to learn, and to gather positive and negative examples illustrating the concepts. As already mentioned, the process of generalization is an inductive one and cannot guarantee the validity of its results. Therefore, we need a fourth step to validate the knowledge that has been learned. We shall not take up this question in this paper.

Actually, the application of ML techniques is not as straightforward as suggested by this scheme. Collecting the examples is an easy task for an expert - in the application to ATC, a month work delivered roughly hundred examples - but building up the BK is a difficult one. Beside the dialogue with the expert, we have to find a language of representation adapted to the domain and to the generalization tool and to translate BK, and positive and negative examples, into this language.

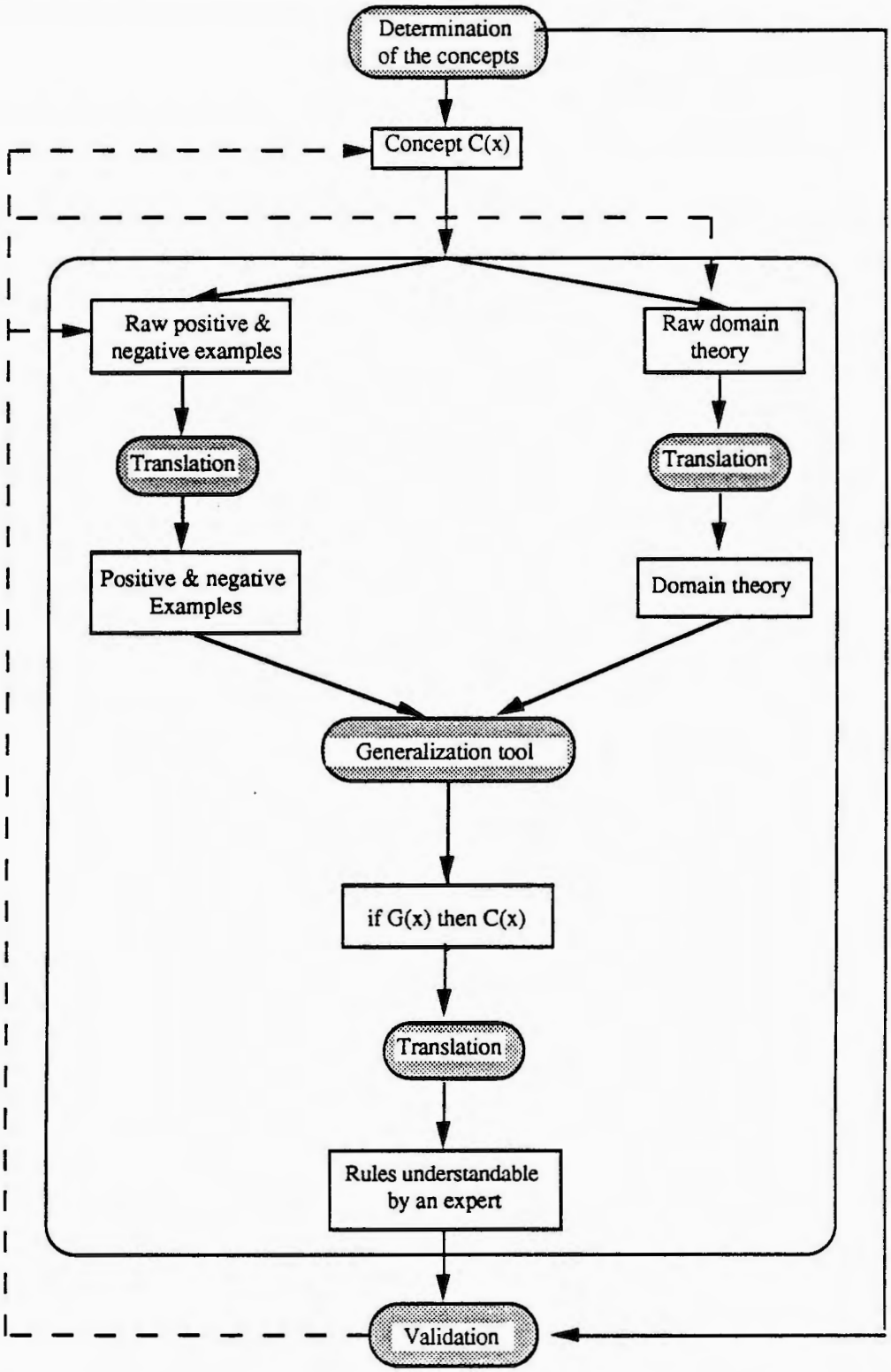


Figure 2. *The acquisition cycle including the steps of translation of knowledge.*

3 - APPLICATION TO ATC

3.1 - Introduction

At the start of the project, we thought that the main problem would be the use of first order knowledge. Actually, our generalization tool is well-adapted to ATC knowledge, and the main problems come from the extraction of BK and, surprisingly, from the definition of the predicates, i.e., the expert shows an unexpected propensity to provide wrong definitions of his own vocabulary.

We describe here the problems that we had to face, and we point out the solutions adopted. For each of them, we will distinguish whether they were inherent to the process of learning expert rules, or to the application of our particular generalization tool. We have been using the new knowledge acquisition scheme, presented in figure 2. This figure shows three main differences with the first one. Transformation into a first order representation of "raw" examples and theory, and back-translation of the learned rules in order to make them understandable to the expert. For the sake of simplicity, we have not mentioned here two steps which are essential and which take place ahead of the learning task, i.e., the choice of the language representation and of the generalization tool.

3.2 - Description of the complete knowledge acquisition task

3.2.1 - Choosing the language representation

This problem can be divided into two linked subproblems,

1- choosing the basic vocabulary,

2 - choosing the formalism, logical, propositional or predicate, attribute-value representation, etc.

In each example, we have at least two objects, the two planes in conflict. It would have been possible to use an attribute-value representation, but we would have to duplicate all the descriptors of the planes. Besides, it may happen that other planes could play a part in the decision of the expert. Moreover, we have many relations between them. The expert will forget some of them, he will represent only the pieces of knowledge that he has in mind at present. This means that we cannot express all the knowledge in the initial representation of the examples, we need theorems that will introduce new pieces of knowledge in the examples. Most of the knowledge of the expert is first order knowledge like for instance the relation between speed, time and distance.

3.2.2 - Choosing the generalization tool

In this application to ATC, we choose the generalization tool OGUST, the main features of which have been presented in section 2.2.. For we have already shown the importance of first order knowledge for ATC. Moreover, we know that in the domain of ATC, there exists a BK. It is not explicit, and it is part of the engineering work to obtain this knowledge from the expert. Finally, we need explanations of the behavior of the system, for at least two reasons,

- we have to validate the learned rules.
- the BK acquired from the expert may contain errors and we have to debug it.

A negative example in this domain would be a crash between two aircraft due to a decision of ATC, and no such negative examples were made available to us. The limitation of OGUST due to its inability to treat negative examples was not a large problem in this application.

3.2.3 - Determining the concepts to learn

The aim of this application is learning decision rules used by a human controller from examples of his behavior when solving a conflict. We have defined six actions that can be performed on

aircraft to avoid an impending accident between them, as for instance a change of direction, which is a temporary modification of the route of the aircraft, or a stabilization of the aircraft to a given height. A solution of a conflict between two planes is the combination of two of these actions, one performed on each aircraft.

For each solution S, we want to learn a description of the situations for which it is advisable to use again this solution. To that purpose, we have to cluster all the examples showing the same solution and to generalize them, using OGUST. The recognition function that we will obtain might be too much general and even perhaps empty. It may happen since OGUST learns a pure conjunctive recognition function of a concept and the concepts defined at first were actually disjunctions of more specific concepts. For instance, the solution *stabilization*, after further discussions with the expert, was decomposed into two kinds of solutions. There are *steady stabilization* of the aircraft to the height it should retain during its flight in the concerned sector[2], or *temporary stabilization* in order to avoid the conflict[3]. Besides, any stabilization can be performed in two ways, either directly or by means of some intermediate levels.
An important task was therefore to define the intermediary concepts to learn.

3.2.4 - Obtaining the examples and rewriting them

Once the language representation and the concepts to learn have been set, the problem is to obtain examples of these concepts and to rewrite them into the desired formalism. In most applications, a questionnaire is established and the expert fills up this questionnaire in real situations. The questionnaire can be viewed as a list of attributes with their possible values, the attributes being determined from the basic vocabulary. For a given situation, the expert checks for each attribute the right value and mentions whether it is a positive or a negative example of the concept. The examples are then translated into the right formalism.

3.2.4.1 - Obtainment of "raw" examples
In our application, we had problems to solve before obtaining usable examples. A questionnaire was set up, but experimentation has shown that it was impossible to fill it up in real situations, so much short is the time between the time a conflict situation occurs, and its solution. The expert prefers to work in two steps.
In the first step, he draws a blueprint representing the conflict situation, and adds to it the convenient verbal information not represented on the drawing (figure 3). In the second step, he transcribes the blueprint and the comments into answers to the questionnaire, where all the numeric values have been translated into symbolic ones (figure 4).
The questionnaire is composed of different kind of attributes.

- Attributes describing the global working environment of the controller, as his *present work load*, his *foreseen work load*, etc.
- Attributes describing the characteristics of each aircraft. For instance, the attribute *type* takes the values *superjet*, *jet*, *turbo* or *propeller*. The controller gives the values of the attributes for each aircraft. A special attribute here is the solution applied to the aircraft to solve the conflict.
- Attributes describing relations between aircraft, which can be divided into two kinds.
  + Attributes describing orientated relations between the two conflicting planes. For instance, the relative position between the two aircraft can take the values *ahead*, *slightly ahead*, *equidistant from conflict occurrence*, *just above each other*.
  + Attributes describing non-orientated relations between the aircraft, as for instance, the *angle of the foreseen routes of the aircraft just before conflict occurrence*, etc.

---

[2] Aircraft controllers work on a well-defined area of space, called a sector. They take in charge a plane entering their sector, and pass along to a colleague when the plane exits their sector.
[3] When controllers recognize that two or more planes are on a possible collision course within their sector, they say that these planes are in a situation of conflict.

Present work load : high steady low
Foreseen work load : high steady low

Estimation of the angle of the routes just before conflict occurrence : 90°	FM1431	FBUYT
Estimation of the angle of the routes after conflict occurrence :	10° D	0°
Relative position of the aircraft : equidistant		
slightly ahead		X
ahead		
doubtful		
Rate of raise : high		
standard	X	
low		
Relative ground speed, relative air speed :	160	190
Qualitative trajectory in vertical plane at the time of entering the sector :		
steady		X
evolutive	↑	
Qualitative trajectory in vertical plane in the sector :		
steady		X
evolutive	↑	
Qualitative trajectory in vertical plane at the time of exiting the sector :		
steady	X	X
evolutive		
Destination : close	X	X
West-European		
far away		
Environmental limitations : active zone		
other flight		
nearby sector		
Parameters describing how much an environmental limitation hinders :		
change of direction to the left, value, distance or beacon		
change of direction to the right, value, distance or beacon		
change of route to the left, value, distance		
change of route to the right, value, distance		
Uncertainty : poor radar contact	X	
poor readibility of radar image	X	
unstable information		
Distance of the flights from the conflict point at the time of actualization of the solution :	20 NM	

Solution : steady stabilization
- temporary stabilization
- change of direction
  - parallelization of routes
  - moving away conflict point
  - bringing closer the conflict point
- change of route
- speed regulation
- height regulation
- left to pilot's estimation

FM 1431 authorized FL 80 (below FBUYT)

Figure 3 (upper part)

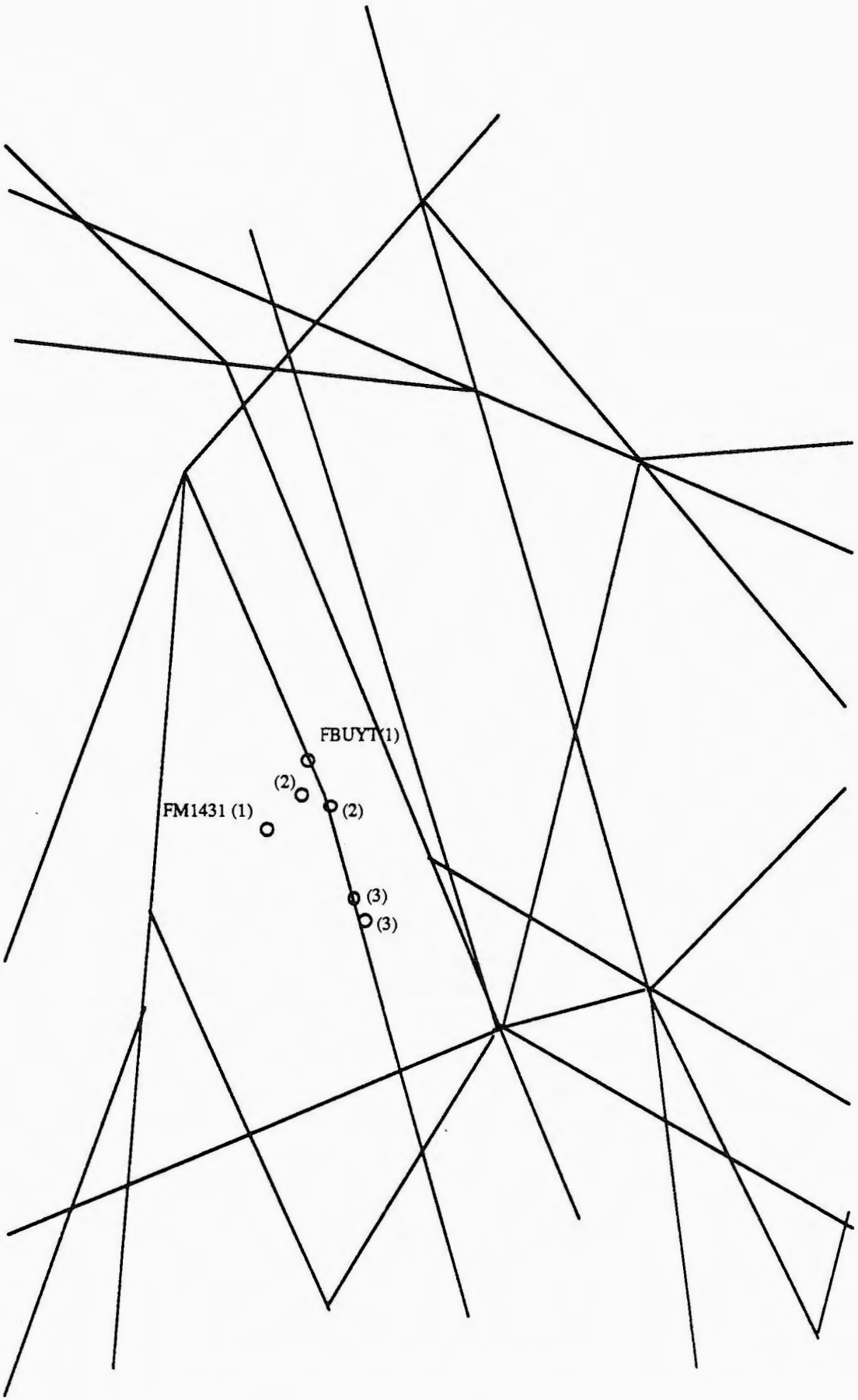


Figure 3 : *A conflict situation written during the control task of the expert*

	FM1431		FBUYT
Present work load	high	steady	low
Foreseen work load	high	steady	low

Estimation of the angle of the routes of the aircraft just before conflict occurrence

- same routes
- sharp convergence
- convergence
- broad convergence
- front encounter

Estimation of the angle of the routes of the aircraft after conflict occurrence

- same routes
- routes branching off

Estimation of the angle between the present route and the route for direct exit

- null	- null
- small change	- small change
- large change	- large change
- left	- left
- right	-right

Relative position of the aircraft as seen from the conflict point

- equidistant

- slightly ahead	- slightly ahead
- ahead	- ahead

- overhead

Rate of raise

- high	- high
- standard	- standard
- low	- low

Relative ground speed

- same

- faster	- faster
- much faster	- much faster

Relative air speed

- same

- faster	- faster
- much faster	- much faster

Type		
	- superjet	- superjet
	- jet	- jet
	- turbo	- turbo
	- propeller	- propeller

Qualitative trajectory in vertical plane at the time of conflict analysis

- steady	- steady
- up	- up
- down	- down

Qualitative trajectory in vertical plane at the time of entering the sector

- steady	- steady
- up	- up
- down	- down

Figure 4 (upper part)

Qualitative trajectory in vertical plane in the sector

- steady	- steady
- up	- up
- down	- down

Qualitative trajectory in vertical plane at the time of exiting the sector

- steady	- steady
- up	- up
- down	- down

Destination

- close	- close
- West-European	- West-European
- far away	- far away

Forbidden Actions

- active zone	- active zone
- other flight	- other flight
- slight change allowed	- slight change allowed
- small change allowed	- small change allowed
- left	- left
- right	- right

Action forbidden unless special authorizing

- nearby sector	- nearby sector
- other flight	- other flight
- slight change allowed	- slight change allowed
- small change allowed	- small change allowed
- left	- left
- right	- right

Uncertainty

- poor radar contact	- poor radar contact
- poor readibility of radar image	- poor readibility of radar image
- unstable information	- unstable information

Distance of the flights from the conflict point at the time of actualization of the solution

- far away
- mean distant
- nearly

Solution

- steady stabilization	- steady stabilization
- temporary stabilization	- temporary stabilization
- change of direction	- change of direction
- change of route	- change of route
- slight	- slight
- small	- small
- left	- left
- right	- right
- parallelization of routes	- parallelization of routes
- moving away conflict point	- moving away conflict point
- bringing closer the conflict point	- bringing closer the conflict point
- crossing behind	- crossing behind
- crossing ahead	- crossing ahead
- speed regulation	- speed regulation
- height regulation	- height regulation
- left to pilot's estimation	- left to pilot's estimation

Figure 4 : *the transcription of the conflict situation of figure 3 into a questionnaire*

The second step is then to translate the questionnaire into a formalism adapted to our generalization tool.

3.2.4.2 - Rewriting the examples in OGUST formalism

The choice of the constants and of the predicates depends on the behavior of the generalization tool and is essential to learn good generalizations. We see at least three ways of transforming a pair *(attribute, value)* into a first order representation but the list is not exhaustive. For more details, see (Vrain and Kodratoff, 1991). Let us illustrate it by the pair (present_work high) that means that the present work load of the controller is high.

- 1st representation: attribute(value), *attribute* becomes a predicate and *value* becomes a constant, as for instance present_work(high);
- 2nd representation: value(attribute), *value* becomes a predicate and *attribute* becomes a constant, as for instance high(present_work).
- 3rd representation: attribute(C) & value(C), *attribute* and *value* become predicates and C is a new symbol for a constant, as for instance present_work(W) & high(W). This representation is close to an object-oriented one. For instance, in our example, we express that there is an object, say *W*, which satisfies the two following properties. *W* represents the present work load of the given situation and *W* is high.This idea of introducing a fictitious object is quite classical. It is usually used to transform facts written in first order logic into structured object representations.

The use of the 1st representation would lead our generalization tool to learn absurd recognition functions, when two different attributes share the same value.
For example, let us suppose that according to it, we represent the case when the present work load of the controller and his foreseen work load are high by *present_work (high) & foreseen_work (high)*.
OGUST is not able to make a difference between the *high* of the present work load and the one of the foreseen work load. They represent for OGUST the same object and they are generalized by the same variable, yielding *present_work (vg1) & foreseen_work (vg1)*. The variable *vg1* represents an object which is together the present work load and the foreseen work load. Of course, such a representation is nonsense since it merges into a unique variable two different objects.

This problem does not appear with the 2nd and the 3rd representations. For instance, with the 3rd representation, the example becomes *present_work (present_work_1) & high (present_work_1) & foreseen_work (foreseen_work_1) & high (foreseen_work_1)*.
The two objects present_work_1 and foreseen_work_1 are considered as distinct, since they do not have the same name.

The 2nd representation could prevent OGUST from using BK correctly, thus leading to over-generalizations.
For example, suppose that in E1 the foreseen work load is high, and that in E2 the present work load is high. The 2nd representation gives :

E1: high(foreseen_work).
E2: high(present_work).

Suppose also that the attributes *present_work* and *foreseen_work* can be generalized into the attribute *amount_work*.
OGUST uses BK on the predicates, but does not on the constants. Therefore, it generalizes the two constants *foreseen_work* and *present_work* into the variable of generalization *vg1* and learns

G: high(vg1).

It does not use the previous piece of knowledge and looses the information that *vg1* is an amount of work.

This problem arises neither with the 1st representation, nor with the 3rd one. Let us consider for instance, the 3rd one. The examples are written

E1: foreseen_work(foreseen_work_1) & high(foreseen_work_1),
E2: present_work (present_work_1) & high (present_work_1),

the constants foreseen_work_1 and present_work_1 are replaced in each example by a variable of generalization, called vg1.

E1: foreseen_work(vg1) & high(vg1) [(= vg1 foreseen_work_1)]
E2: present_work(vg1) & high(vg1) [(= vg1 present_work_1)]

OGUST uses its BK to suppress the discriminating occurrences of the variable *vg1* in the examples and we obtain the generalization

G: amount_work(vg1) & high(vg1).

The only representation that prevents these two problems is the 3rd one, this is why we have been using it. For instance, to express that in a given example the meeting angle of the routes of the two aircraft before the conflict point is minimal, we write

angle_before (ANGLE_BEFORE_1) & minimal (ANGLE_BEFORE_1).

If the solution to the conflict consists in changing the course of the first aircraft to the right, we shall write

solution (AIRCRAFT_1 SOLUTION_1) & change_course (SOLUTION_1)
& to_the_right (SOLUTION_1)

The information that the type of the second aircraft is a jet is written

type(AIRCRAFT_2 TYPE_2) & jet (TYPE_2).

The representation chosen is a structured-object-oriented one, written in first order logic. We deal with some basic objects as the present work load, etc. and we have structured objects, the aircraft, composed of basic objects as the type, the air speed, ... Moreover, all the objects are implicitly typed, so that we cannot match for instance the present work load and the foreseen one. An algorithm has been implemented to translate automatically the examples written in terms of the questionnaire in this formalism.

3.2.5 - Rewriting BK as Horn clauses

3.2.5.1 -The choice for the representation of the examples has a great influence on the writing of BK. For instance, let us suppose that we want to express the following knowledge.

IF a given plane has a steady profile when entering, exiting the sector and inside the sector THEN it is on cruise.

The flight profile of a plane, within the sector, when entering it, and when exiting it are attributes given in the questionnaire. On the contrary, the characteristics of the flight, that is to say *on cruise* is a new one. To be coherent with our structured representation, we must define a new attribute characterizing the flight and which can take three values *cruising, moving from cruise state, partially moving out of cruise state*. Let us call this new attribute *flight*. The theorem is then written.

TH1:

$\forall$*x $\forall$*prof_ent $\forall$*prof_in $\forall$*prof_out [[profile_enter(*x *prof_ent) & steady(*prof_ent) & profile_in (*x *prof_in) & steady (*prof_in) & profile_out (*x *prof_out) & steady (*prof_out)]

$\Rightarrow$ $\exists$*fl [flight(*x *fl) & cruising(*fl)]]

The conclusion of the theorem expresses that there exists an object which represents the flight of *x and which satisfies the property of being "cruising".

3.2.5.2 - Theorems must be expressed as Horn's clauses, i.e., the conclusion of a theorem must be a single atom and we do not allow existential variables.

To solve these problems, we replace the previous theorem by the two following ones, the existential variable *fl has been skolemized, introducing a new symbol of function f.

TH21:

∀*x ∀*prof_ent ∀*prof_in ∀*prof_out [profile_enter(*x *prof_ent) & steady(*prof_ent) & profile_in (*x *prof_in) & steady (*prof_in) & profile_out (*x *prof_out) & steady (*prof_out)

⇒ flight(*x f(*x *prof_ent *prof_in *prof_out))]

TH22:

∀*x ∀*prof_ent ∀*prof_in ∀*prof_out ∀*fl [profile_enter(*x *prof_ent) & steady(*prof_ent) & profile_in (*x *prof_in) & steady (*prof_in) & profile_out (*x *prof_out) & steady (*prof_out)] & flight(*x *fl)

⇒ cruising(*fl)]

The two formulas are in general not equivalent (TH1 is a logical consequence of TH21 & TH22 but the reverse is not true) and we have to check that the new formula, TH21 & TH22, is still valid.

3.2.5.3 - OGUST does not deal with functions and the application of a theorem on an example can, as we have seen, introduce a functional term. In order to treat this case, we replace the functional symbol by its value for the specific example to which the theorem is applied. We thus replace the functional symbol by a new constant.

3.2.6 - Modification of the algorithm

In OGUST, a constant represents an object and we look for the most similar objects, in order to find their common properties. In the application to ATC, the only real objects we have are the aircraft but, as seen in section 3.2.4, we introduce fictitious objects to represent the examples. We can single out two among these fictitious objects,

- the objects describing the global environment and independent of the other objects, such as the present work-load of the controller,
- the objects representing properties of the aircraft, such as their types, their profiles.

To compute the similarity between two aircraft, we must take into account the common properties related to the last kind of fictitious objects.

To take into account this notion of structured objects, we have modified the function "sim", computing the similarity between two objects in the following way.

- For two basic objects O1 and O2 of the same type

new_sim (O1, O2) = sim (O1, O2),

- For two structured objects O1 and O2 of the same type and composed of the objects (O11,..., O1n) and (O21,..., O2n),

$$\text{new_sim}(O1, O2) = \Sigma_i \, \text{sim}(O1i, O2i).$$

Once the constants to match have been chosen, we replace the two constants by a variable *vg* and we have to put them into structural matching. Two cases are still possible.

- The two constants O1, O2 represent basic objects, and the structural matching starts at once.
- The two constants O1, O2 represent structured objects. The structural matching takes place on the basic objects that are composed of. Let us call (O11,..., O1n) and (O21,..., O2n) the basic objects composing O1 and O2. For each pair of objects (O1j, O2j) of the same type, we introduce a new variable *vgj* and we put them into structural matching.

3.2.7 - Rewriting the generalization

The formalism used to represent the learned rules is not natural for the expert. In order to show them to the expert, we are forced to translate the concepts learned by OGUST in a representation more familiar to him. The representation we use is the one of the questionnaire.

A module has been implemented to make this step automatic.

4 - CONCLUSION

We have been showing that knowledge intensive generalization techniques need even more than a generalization tool making use of the field expert's knowledge, the so-called background knowledge (BK). The very use of BK entails a first order logic representation, which is not friendly to the expert. Thus, besides the generalization tools, we need other tools to perform the transformation of the examples and the expert's knowledge in first order, and, once the learning as been taking place, we need translate it back into a representation that is acceptable to the expert. All these translation steps are far from trivial, and they contain choices that must be carefully discussed with the expert since they can strongly biase the learning.
The success of the learning process often relies on a proper choice of the formalism used to represent knowledge, and this includes the choice of the basic vocabulary. It is obvious that if a concept depends on a factor and if we have no way to represent this factor with the primitives of the language we have defined, we shall not be able to learn a good definition of the concept. The choice of the primitives of the language is very much domain dependent, so that we have no hope of automatizing this step and we can only get them by means of a dialog with the expert. The latter will certainly be improved by using techniques stemming from knowledge acquisition. Presently, we study the use of knowledge acquisition tools to elicit this basic vocabulary. In the application to ATC, this choice has led to the elaboration of a questionnaire containing those language primitives which seemed important for the decision of the controller. We had also to choose a logical formalism, and we chose first order formalism against a more classical attribute-value representation. The expressive power of first order logic is higher that the one of an attribute-value representation, but most existing systems did not choose it because of the lack of efficiency of the processes that use it. We have shown that in some cases, as in ATC, we are compelled to use it. Once first order is chosen, we have been showing that we have to determine which primitives of the language will become predicates, and which one must play the role of constants. In our application, our choice led to simulate an object-oriented representation and we had to elaborate translation modules.
Another task is the obtainment of the positive and negative examples and their translation into the formalism we have chosen. For ATC, this task can itself be divided into several steps. The expert in real situations has no time to fill the pre-determined questionnaire. Therefore, he writes only the information which is necessary to remember the conflict, and the way it has been solved. Afterwards, he transcribes them into answers to a questionnaire. A friendly interface enables to store the answers into the BK base, and a module translates them from their attribute-value representation into the first order one.

The most difficult problem is certainly the obtainment of this BK base. We did it by means of discussions with the expert, but we are conscious that it was far from being satisfactory. When expressing *if ... then ...* rules, the expert forgets very often some conditions that make the rules false. Moreover, he does not give all the rules, and we thus obtain an incomplete BK. The falsity of some pieces of knowledge leads to erroneous generalizations, and the lack of pieces of knowledge leads to over-generalizations. The validation step, by the field expert, of the learned rules is therefore of major importance. When an expert rejects a rule, we have to find where the errors are, in the definition of the concept, in the examples, or in the BK. We plan to use the explanations provided by the generalization tool OGUST in order to help the expert to detect where the errors occur and to correct them.

Another problem, which has not been treated here, is the one of numeric values. This application shows the necessity of introducing numeric values into the generalization tool and leads to the development of KBG (Bisson, 1990).

Among all the tasks that we had to solve during this application to air traffic control, the ones implemented, that is, the generalization tool, the interface to stock the examples into an attribute_value representation, and the translation modules have been written in Lelisp, version

References

Avizienis A., Ball D.E. (1987). "On the Achievement of a Highly Dependable and Fault-Tolerant Air Traffic Control System", *Computer*, February 1987, Volume 20, Number 2, pp., 1984-92.

Bisson G. (1990). "KBG: A Knowledge Based Generalizer", proceedings of the Seventh International Conference on Machine Learning, University of Texas, Austin, June 21-23 1990, pp. 9-15.

Bleistein S., Goettge R., Petroski F., Wiseman R. (1987). "Capacity Management of Air Traffic Control Computer Systems", *Computer*, February 1987, Volume 20, Number 2, pp. 73-83.

Cannat J.J., Vrain C.(1988). "Machine Learning applied to air traffic control", Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, pp. 265-274.

Chaib-Draa B., Mandiau R., Millot P. (1988). "CIS: Cooperating Intelligent System", Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, pp. 315-329.

Gosling G.D., Hockaday S.L.M. (1984). "Identification and Assessment of Artificial Intelligence Techniques in Air Traffic Control", Research Report, UCB-ITS-RR-84-14, Institute of Transportation Studies, University of California, Berkeley.

Hunt V.R., Kloster G.V. (1987). "The FAA's Advanced Automation System: Strategies for Future Air traffic Control Systems", *Computer*, February 1987, Volume 20, Number 2, pp. 19-32.

Kodratoff Y. (1983). "Generalizing and particularizing as the techniques of learning", *Computers and Artificial Intelligence 2*, 1983, pp. 417-441.

Kodratoff, Y., Ganascia, J. G., Clavieras, B., Bollinger T., Tecuci G. (1984). "Careful generalization for concept learning", Proceedings ECAI-84, Pisa, 1984, pp. 483-492.

Kodratoff Y., Ganacia J. G. (1986). "Improving the Generalization step in Learning", in *Machine Learning: An Artificial Intelligence Approach,* Vol.2, Michalski R.S., Carbonell J.G. and Mitchell T.M. (Eds), Morgan Kaufmann Publishers, 1986, pp. 215-244.

Kodratoff Y., Tecuci G. (1988). "Learning based on conceptual distance", *IEEE Trans. on Pattern Anal. and Mach. Int. 10,* 1988, pp. 897-909.

Kodratoff Y.(1988). *Introduction to Machine Learning*, Pitman Publisher, 1988.

Kodratoff Y. (1990). "Learning expert knowledge by improving the explanations provided by the system" in *Machine Learning: An Artificial Intelligence Approach, Volume III*, Y. Kodratoff, R.S. Michalski (Eds.), Morgan Kaufmann, San Mateo CA, 1990, pp. 433-465.

Lebowitz M. (1990). "The utility of Similarity-based Learning in a World Needing Explanation", *Machine Learning, an Artificial Intelligence Approach, volume III,* Eds.: Y. Kodratoff and R.S. Michalski, Morgan Kaufman Publishers, pp. 399-432.

Levi R., Shalin V.L., Wisniewski E.J., Scott P.D.(1987). "An Analysis of Machine Learning

Applications for Pilot Aiding Expert Systems", Interim Report for period October 1986 - July 1987, Contract N° F33615-86-1125, Honeywell Systems and Research Center, Minneapolis.

Michalski R.S. (1983) "A Theory and methodology of Inductive Learning", *Artificial Intelligence, Vol.20, No.2, feb. 1983.*

Michalski R.S., Kodratoff Y. (1990). "Research in Machine Learning; Recent Progress, Classification of Methods, and Future Directions", in *Machine Learning: An artificial intelligence approach*, Vol. 3, Kodratoff Y. & Michalski R. S., (Eds.), Morgan Kaufmann, 1990, pp. 3-30.

Onken R.(1988). " Automatic pilot aid for situation awareness and flight planning", Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, pp. 125-137.

Phillips M.D. (1987). "The Quantification of Operational Suitability", *Computer,* February 1987, Volume 20, Number 2, pp. 63-72.

Planchon P., Berrada J. (1988). "ANTICO: an anti-collision system in air traffic control", Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, pp. 331-346.

Speyer J.J. (1988). "Towards design-induced error tolerance", Colloque International Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace, Toulouse, France, 28-30 septembre 1988, pp. 69-94.

Stepp R.E., Michalski R.S. (1983). "Learning From Observation: Conceptual Clustering", *Machine Learning, an Artificial Intelligence Approach,* Michalski R.S., Carbonell J.G., Mitchell T.M., eds. Morgan-Kaufmann 1983, pp. 331-363.

Vrain C. (1987). "Un outil pour la Généralisation Utilisant Systématiquement les Théorèmes: le système OGUST", Thèse de troisième cycle, L.R.I. Université de Paris-Sud, Orsay, France, 25 Février 1987.

Vrain C. and Kodratoff Y. (1989). "The use of Analogy in incremental SBL", *Knowledge Representation and Organization in Machine Learning*, eds. K. Morik, Lecture Notes in Artificial Intelligence, Springer Verlag n° 347.

Vrain C. (1990). "OGUST: A System that Learns Using Domain Properties Expressed as Theorems", *Machine Learning, an Artificial Intelligence Approach,* volume III, eds.: Y. Kodratoff, and R. S. Michalski, Morgan Kaufman Publishers, pp. 360-382.

Vrain C. and Kodratoff Y. (1991). "Acquisition of first order knowledge and its application to air traffic control", proceedings of the Fifth European Knowledge Acquisition for Knowledge Based Systems Workshop, EKAW 91, Crieff Hydro, Scotland, May 20th - May 24th 1991.

Einstein—An interactive inductive knowledge acquisition tool

Geoffrey I. Webb
Department of Computing and Mathematics
Deakin University
Geelong 3217 Australia.
webb@deakin.oz.au

ABSTRACT

Einstein is a knowledge acquisition system that incorporates data-driven inductive rule development and refinement in a user driven production rule development and evaluation environment. This allows the user and the induction system to interact as a cooperative knowledge acquisition team. Unique features of this system include efficient automated inductive refinement of existing production rules, interactive user management of machine learning facilities, including local and global guidance, interactive specification of key examples and counter-examples and interactive case-based rule assessment.

INTRODUCTION

Einstein supports cooperative development of rule based expert systems through interactive collaboration between a domain expert and a *data driven machine learning* (DDML) sub-system.

Given a set of examples of a classification task (the *training set*), DDML, without access to any other information, develops a general procedure for classifying cases from the domain. During the induction task, such a system does not have available, and thus is unable to take into account, general epistemological and domain specific knowledge. In view of this, it is often the case that the output of an autonomous DDML system should only be treated as a 'first draft' of a knowledge base. In most cases, before an expert system created by an autonomous DDML system will be suitable for real world application it will require considerable refinement by a knowledge engineer and/or a domain expert. This process is both slow and costly.

Numerous approaches have been developed for automated knowledge capture through interactive interview with an expert. A number of these support the use of machine learning techniques to refine the knowledge so obtained (see, for example, Boose, Shema & Bradshaw, 1989; Gaines & Rappaport, 1989.)

Einstein implements another alternative, *collaborative knowledge acquisition.* This method enables interactive collaboration between a domain expert and a DDML system, combining the demonstrated power of both knowledge acquisition methodologies. At any stage, either the induction system or the user may further refine the existing set of production rules.

A number of researchers (see, for example, Shapiro, 1987 and O'Niel & Pearson, 1987) have used this approach in the context of induction of decision trees. These systems have allowed the domain expert to interactively provide guidance to the induction system and to evaluate and modify the trees that are produced. Collaborative knowledge acquisition has also been used in the acquisition of exemplars (Bareiss, 1989.)

However, this approach previously has not been utilized in the context of the induction of production rules. Einstein provides collaborative knowledge acquisition in the context of induction of production rules by generalization, using DLG (Webb, 1990), a variant of Michalski's (1984) Aq algorithm.

There are several advantages associated with the use of production rules, as opposed to decision trees, in collaborative knowledge acquisition.

- It is widely accepted that production rules are a more natural formalism for human experts to manipulate than decision trees. (See, for example, Quinlan, 1987.)
- As most existing expert system technology is based on production rules, more powerful tools are available for supporting expert systems expressed as production rules than are available for supporting those expressed as decision trees.
- The process can be used to refine existing expert systems, most of which are expressed as production rules.

The application of collaborative knowledge acquisition in the context of the induction of production rules by generalization is achieved by successive cycles of refinement to a rule set. The interaction alternates between the induction system performing data driven refinement and the domain expert performing model driven refinement. The domain expert also provides strategic and tactical direction for the induction process. By these means, the user is able to provide model driven direction to the data driven induction and perform model driven refinement of the resulting rule set.

The first draft of the rule set can be induced by the induction system, created by the domain expert, or taken from an existing expert system. The actions available to the domain expert include—

- evaluating the accuracy of production rules with respect to a set of examples;
- proposing new production rules;
- modifying existing production rules;
- deleting existing production rules;
- providing tactical and strategic directions to the induction system;
- interactive generation of significant examples and counter-examples; and
- control of the machine learning techniques to be applied in the induction process.

Einstein has been implemented on Apple Macintosh computers so as to take advantage of the direct manipulation interface that the Macintosh provides.

DLG

Einstein uses the DLG DDML algorithm (Webb, 1990.) DLG combines the least generalization (Plotkin, 1970, 1971) and Aq (Michalski, 1984) DDML algorithms to provide computationally efficient induction of disjunctive class descriptions. These are expressed to

the user in the form of production rules. Each disjunct of a class description is presented as a separate rule. DLG also supports the revision of existing rules (Webb, 1991b).

DLG can be described as follows.

Input: *POS* (a training set of instances belonging to the class of interest)
NEG (a training set of instances not belonging to the class of interest)

Output: *R* (a set of production rules for the class.)

initialize *R* to $\varnothing$.
while *POS* is not empty
randomly select and remove an instance *i* from *POS*.
initialize *c* to the most specific rule that covers *i*.
for *x* is set to each successive instance in *POS*
set *l* to a least generalization of *c* that covers *x*.
if *l* does not cover any instance in *NEG*
set *c* to *l*.
remove from *POS* all instances covered by *c*.
add *c* to *R*.
end while.

The computational efficiency of this algorithm is important in the context of interactive knowledge acquisition as low computational complexity minimizes the delay experienced by the user while induction is performed.

The worst case computational complexity of DLG can be calculated as follows.

Let P = the number of examples in POS, N = the number of examples in NEG and A = the number of attributes in the domain.

The worst case for DLG occurs when a separate rule is required to cover each positive example, that is, when it is not possible to create rules that cover multiple positive examples and do not cover negative examples. In this case, the rule generalization process will occur once for each case in POS, and when a rule has been developed, the size of POS will decrease by the one case that is covered by the new rule. Each rule will be generalized against every example remaining in POS, thus $\sum_{p=1}^{P} p$ generalized rules will be generated. Each of these rules will be tested to see if they cover any negative examples. In the worst case, they will always cover the last negative example only. As a consequence, $\sum_{p=1}^{P} pN$ rule tests will be performed. Each of these rule tests requires a test on every attribute, resulting in an overall complexity of the order $O\left(\sum_{p=1}^{P} pNA\right)$.

By contrast, the worst case computational complexity of a standard implementation of Aq (using here the algorithm and terminology presented by Michalski, 1984) can be calculated as follows. Note that the terms 'complex' and 'selector' used by Michalski are equivalent respectively to the terms 'rule' and 'clause' used herein. Note also, that only the specialization

stage of standard Aq (steps 4 and 5) is examined here. The complexity of the generalization stage is far more difficult to present due to its reliance on the nature of the output from the specialization stage. Note also that, for ease of analysis, it is assumed that the specialization stage will terminate when and only when the first consistent solution is found. In practice, the algorithm may terminate before the analysis below suggests, due to exceeding a predefined maximum time allocation, in which case a poor solution will be found, or continue for longer, trying to find additional solutions.

Let P = the number of examples in POS, N = the number of examples in NEG, A = the number of attributes in the domain, W = the beam width and B = the branching factor ($0 < B \leq 1$).

Like DLG the worst case occurs when a separate complex (rule) is required to cover each positive example. Ignoring the additional complexity introduced by constructive generalization, in the worst case, each complex will require A selectors, taking A cycles through the specialization step (Step 5). Thus, in total, there will be PA specialization steps. In each of the specialization steps, the beam will contain W complexes, each of which will be conjoined with selectors for B of the lower ranked attributes (on average, 0.5 of the remaining attributes will be ranked lower than a particular attribute.) As each specialization step adds a selector for one attribute, $P\sum_{a=1}^{A}.5aBW$ candidate complexes will be generated in total. Each of these must be evaluated, requiring a test against every example in NEG and every example remaining in POS, a total of $\sum_{p=1}^{P}\sum_{a=1}^{A}.5aBW(p+N)$ tests on complexes. Each of these tests on a complex requires a tests on selectors, resulting in a total complexity of the order

$$O\left(\sum_{p=1}^{P}\sum_{a=1}^{A}.5aBW(p+N)a\right) \text{ or } O\left(\sum_{p=1}^{P}\sum_{a=1}^{A}.5a^2BW(p+N)\right).$$

To provide measures that are less difficult to compare, it is useful to consider favorable cases. In the most favorable case for standard Aq, one rule (complex) containing only one clause (selector) can cover all positive cases and no negative case.

In this circumstance, for DLG, the rule generalization process will occur once and the size of POS will be P. P generalized rules will be generated. Each of these rules will be tested to see if they cover any negative examples, which they will not, requiring that every example in NEG be examined. As a consequence, PN rule tests will be performed. Each of these rule tests requires a test on every attribute, resulting in an overall complexity of the order O(PNA).

For standard Aq, the one complex will require only one selector, taking a total of one specialization step. In this step, the beam will contain W complexes, each of which will be conjoined with selectors for B of the lower ranked attributes (on average, 0.5 of the $A-1$ attributes remaining.) So, $WB(A-1)$ candidate complexes will be generated in total. Each of these must be evaluated, requiring a test against every example in NEG and every example in POS, a total of $WB(A-1)(P+N)$ tests on complexes. Each of these tests on a complex requires a test on only one selector, resulting in a total complexity for the specialization step of the order $O(WB(A-1)(P+N))$.

By contrast, in the most favorable simple case for DLG one rule will be able to cover all positive cases and no negative case only if it contains A clauses (selectors) each specifying one value. In this case, every positive example will be identical, and so, no generalization will occur, meaning that only one rule need be evaluated. Therefore, the complexity is that of testing a single rule against every negative example, $O(NA)$.

For standard Aq, the one complex will require A selectors, taking a total of A specialization steps. In each step, the beam will contain W complexes, each of which will be conjoined with selectors for B of the lower ranked attributes (on average, 0.5 of those attributes remaining.) As each specialization step adds a selector for one attribute, $\sum_{a=1}^{A} .5aBW$ candidate complexes will be generated in total. Each of these must be evaluated, requiring a test against every example in NEG and every example in POS, a total of $\sum_{a=1}^{A} .5aBW(P+N)$ tests on complexes. Each of these tests on a complex requires a tests on selectors, resulting in a total complexity of the order

$$O\left(\sum_{a=1}^{A} .5aBW(P+N)a\right) \text{ or } O\left(\sum_{a=1}^{A} .5a^2BW(P+N)\right).$$

Further, DLG is guaranteed to find the single rule in both these latter cases (Webb, 1991a), but standard Aq is only guaranteed to find it in the last case if $W = A$ and no time limit is imposed.

Unlike most other variants of Aq which are restricted to discrete attributes, DGL can process both discrete and continuous attributes. The only other member of the Aq family that can process continuous attributes is CAQ, which uses the same general strategies as standard Aq (Whitehall, Lu & Stepp, 1990).

USING EINSTEIN

An Einstein project usually starts with the user compiling a data set of example cases from the domain. These cases must be described in terms of a number of attributes and a classification (called the *decision.*)

The cases may already exist in the form of a database, or may be collected expressly for this purpose. The data is entered into a text file. The format requirements are very flexible. As a result, this file could be generated directly by most database management systems from existing databases. The file is prefaced with a description of the attributes and classes that are to be examined.

Appendix 1 shows the start of a file of example cases for the domain of the diagnosis of Immunoglobulin A Nephropathy. Note that a number of attributes are marked as ignored. These are attributes which are not to be taken into account in developing rules. The three ignored attributes in this data set are used for identification of records.

Einstein supports the use of two types of missing information. These are called unknown and unobtainable values. Unobtainable values indicate that a value is missing and that this is a significant piece of information which the system should take into account when developing rules. Rules may have clauses that require the value for an attribute to be unobtainable. Such a rule indicates that if the value for that attribute is unobtainable then the other clauses in the rule should be used to determine whether the conclusion holds. For nominal attributes, unobtainable is simply an additional value, given identical treatment to all other values. For

ordinal attributes, unobtainable is a different type of value to the other values for the attribute. Effectively, unobtainable is nominal while all other values for the attribute are ordinal.

By contrast, unknown values indicate that the value of an attribute is not known, but that this is not a significant piece of information to be used by the system. Only attribute values that are not unknown are used in induction. Rules cannot incorporate clauses that require the value of an attribute to be unknown.

Having created the file of example cases, the user then runs Einstein and reads in the file. He can now either create a set of rules himself, or request Einstein to induce them.

INTERACTING WITH EINSTEIN

The production rule format that is supported allows a condition consisting of a conjunction of clauses, with at most one clause relating to each attribute, and a single conclusion. The conclusion states that the case belongs to a specific class. The clauses in the condition take different forms depending upon whether the relevant attribute is nominal or ordinal. For nominal attributes, the clause lists a specific set of values for the attribute. Unobtainable is treated as a distinct value. For ordinal attributes, the clause either takes the form of a range specification, or a requirement that the value be unobtainable.

Figure 1 shows a window of rules created by Einstein from the data set shown in part in Appendix A. Note that only three rules are currently in view in the window. Further rules, for the negative class, descend below the area displayed. The number in brackets at the end of each rule represents the number of positive cases covered by the rule.

Rules may be edited by pointing at the rule or clause within a rule, and specifying by menu selection the action to perform. Actions pertaining to entire rules include—

- Adding a new clause. A sub-menu of all attributes for which the current rule does not contain a clause is displayed from which the user can select the attribute to which the new clause will refer. If the attribute is ordinal a dialog allows the user to select the minimum and maximum values that the clause covers or to specify that the value must be unobtainable. Otherwise, if the attribute is nominal, a dialog allows the user to select the values that the are to be covered. A new clause for that attribute is added to the current rule.

- Adding a new rule. A sub-menu of all classes is displayed from which the user can select the class for the new rule. A new rule for that class, with no clauses, is added to the set of current rules.

- Duplicating a rule. A new rule that is identical to the current rule is added to the rule set.

- Setting the status of the rule. A rule may be either accepted, rejected, or revisable. An accepted rule may not be altered by the induction system. A rejected rule is deleted when the induction system is applied to the rule set and ignored when the rule set is applied to a case. A revisable rule can be revised by the induction system.

Actions pertaining to single clauses include—

- Removing the clause from the rule.

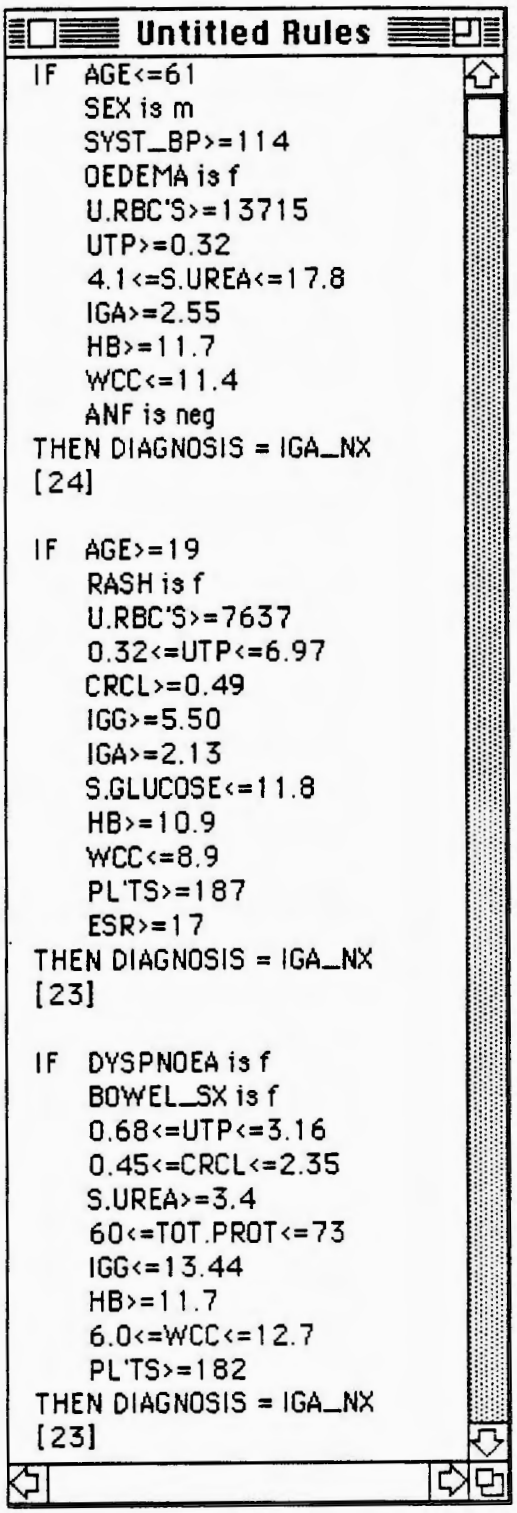


Figure 1: A window containing induced rules.

- Changing the value(s) specified in the clause.

  If the attribute to which the clause refers is ordinal, a dialog allows the user to enter new minimum and maximum values or to specify that the value be unobtainable. To specify no minimum or maximum, the user leaves the appropriate field blank.

  If the attribute is a binary nominal attribute, the value is changed to the opposite value.

  If the attribute is any other nominal attribute, a dialog displays a list of all possible values. The current values are ticked. By clicking onto a value the user ticks or un-ticks it. When all of the values that the user wants selected for the clause are ticked, he clicks on the OK button.

Figure 2 shows an example of rule editing in progress. The user has accepted the second rule which is consequently displayed in bold type. The third rule has been rejected and thus is displayed in italic type. The user is now in the process of adding a clause to the first rule. A dialog will appear prompting for minimum and maximum allowable values for the specified attribute.

Having modified the rule set, in the manner demonstrated above, the user can request that the rules, either for a single class or for all classes, be revised. In this case, the DLG algorithm is applied to the appropriate rules, resulting in data driven inductive rule refinement.

CASE-BASED RULE ASSESSMENT

Einstein supports several features that enable the user to perform case-based rule assessment—that is, to use the example cases to explore and evaluate the quality of the rules that are under development.

One of the ultimate tests of rule quality is how well the rule set performs when applied to new cases. To this end, Einstein allows the user to randomly divide the example cases into two sets—the training set and the evaluation set. Only cases in the training set are used during rule induction. At any stage, the rule set can be applied to the evaluation set, the results of which are then summarized and displayed in two windows. The first of these windows presents a matrix indexed in one dimension by the class assigned to a case and in the other dimension by the correct class for the case. In each cell, the number of cases falling into the cell is displayed. The second summary window displays a table describing the number of cases for each class and how many were correctly classified as well as the number of cases that were assigned to the class and how many of those were correctly classified. This second window is particularly useful when there are a large number of classes.

The rules in Figure 1 were developed with 80% of the data set assigned to the training set. Figure 3 displays the summary windows when those rules were evaluated.

Further analysis of the rule set is supported by a number of windows that each display selected cases from the training and/or evaluation sets. Exactly which sets the cases are selected from can be controlled by the user.

The first window displays all of the selected set of example cases. The user can browse through these cases and modify them as desired. Full case editing facilities are also provided for all of the windows that display example cases described below.

The second window displays all of the positive examples of the current rule. These are the cases that the rule correctly identifies as belonging to its class.

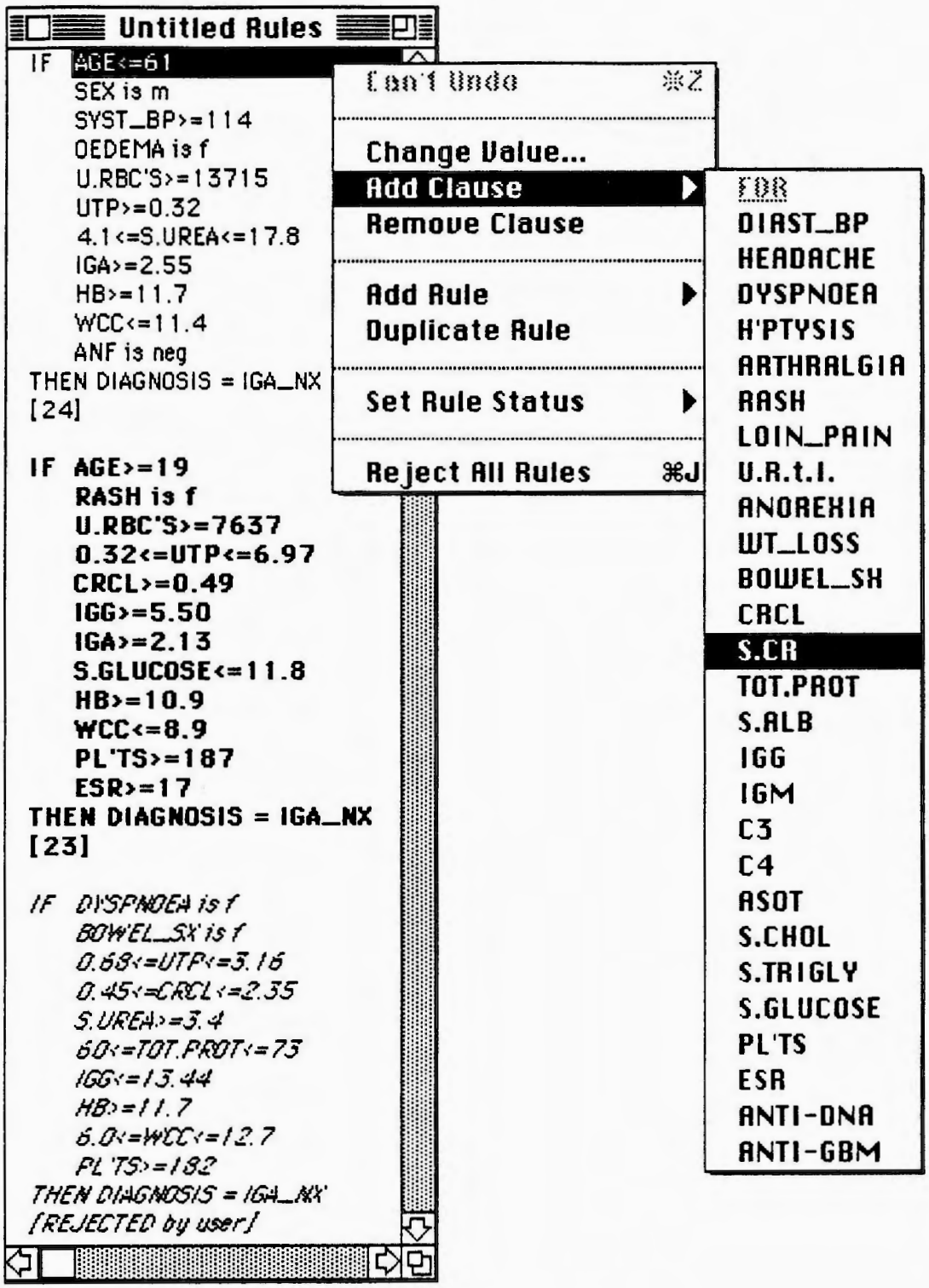


Figure 2: Editing the rule set.

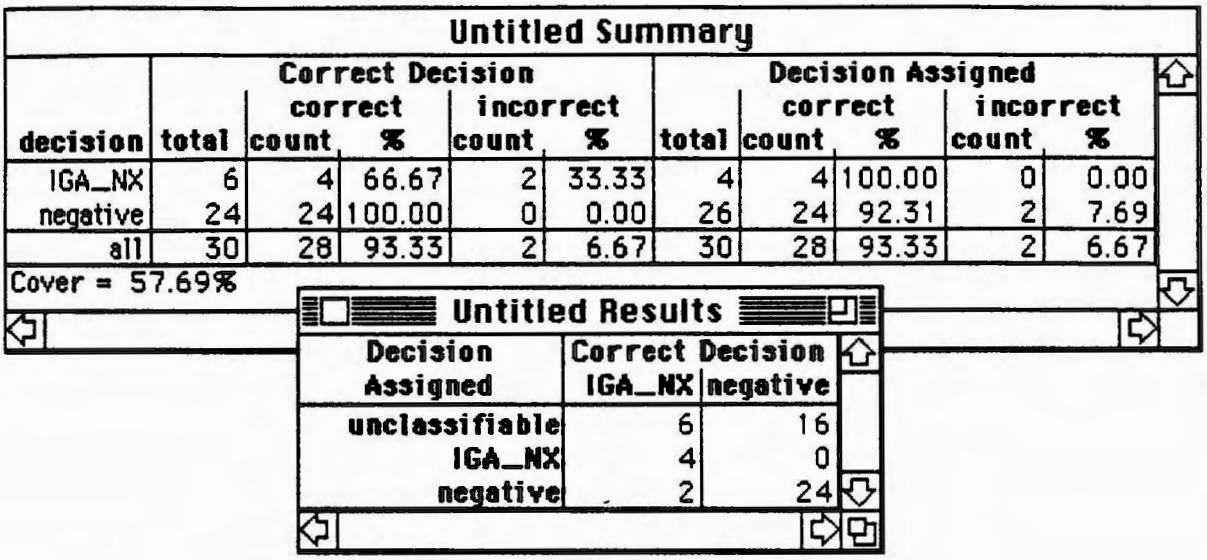


Figure 3: Rule set evaluation summary windows

The next window displays all of the counter-examples to the current rule. These are the cases that the rule incorrectly identifies as belonging to its class.

Another window displays all of the uncovered examples for the current rule. These are cases which belong to the class for the rule, but which are not covered by the rule.

A further window displays all of the insufficient evidence examples for the current rule. These are cases which belong to the class for the rule, but for which it is not possible to determine whether or not the rule applies because values are missing for relevant attributes.

The final examples window displays any indistinguishable examples in the training set. These are cases that cannot be distinguished from other cases in the training set that belong to different classes. Two cases belonging to different classes cannot be distinguished if the respective values are identical for all of the attributes for which values are defined in both cases. Unlike the other example windows, this last window always applies only to the training set.

These windows can be used for a wide variety of purposes. For example, to determine whether a clause in a rule is redundant, and if not, why not, it is only necessary to delete the clause and then view the cases that are added to the counter examples window.

The positive examples and uncovered examples windows provide a good measure of how complete a rule is while the counter examples window provides a good measure of how accurate it is.

If a rule is suspected of being too specific, relevant generalizations will often be suggested by viewing the uncovered examples window. Each case in this window will suggest potential generalizations with the attribute values that are not covered by clauses in the rule.

In a similar vein, if a rule is suspected of being too general, relevant specializations will often be suggested by viewing the counter examples window.

The insufficient information window provides a measure of the need to build redundancy into the knowledge base to accommodate missing values.

Finally, the indistinguishable examples window provides an indication of whether the domain model is sufficient. If cases that do not contain missing values appear in the indistinguishable examples window, either the values of one such case must be incorrect or it must be impossible to correctly classify cases from the domain on the basis of the attributes that are provided.

INTERACTIVE SPECIFICATION OF EXAMPLES

An important facility provided by the system is the ability to provide additional example cases. When developing an expert system, it will often be the case that the domain expert will believe that a particular rule is not correct but not be able to specify exactly how it should be altered. As an alternative, the interactive ability to provide example cases allows the user to demonstrate to the system that a rule is wrong by presenting counter-examples.

Coupled with Einstein's rapid inductive rule refinement, the interactive specification of counter-examples provides a powerful knowledge-acquisition mechanism. The user can remedy deficiencies in a rule set by providing relevant counter-examples and then requesting that the system refine the rules.

Counter-examples may be used either to demonstrate that a specific rule is too general or that the set of rules for a class is too specific. The former is achieved by creating a case that is covered by the rule while belonging to another class. The latter is achieved by creating a case that belongs to the class in question and is not covered by any of the rules.

USING THE RULES

Another facility available to the user is interactive application of the rule set. An interactive expert system shell is supported that allows the user to apply the rule set to arbitrary cases. As each rule is applied to the current case, it is highlighted in the rules window, allowing the user to observe how each rule contributes to the expert system's performance.

At any stage, the user can save the current rule set as a stand alone expert system application for the Macintosh, as OPS5 source code, or in a textual representation.

INDUCTION OPTIONS

Before using the induction sub-system, it is possible to set a number of parameters that alter its performance.

Two styles of production rule are supported. Absolute rules are independent of each other. That is, each absolute rule states that if the conditions hold then the conclusion holds, irrespective of other rules in the rule set. By contrast, relative rules can only be considered in relation to the rules that precede them in the rule set. Each relative rule is interpreted as stating that if none of the preceding rules applies and the current rule's conditions are satisfied, then the conclusion obtains. All rules in a rule set must be of the one type—either absolute or relative.

Einstein supports two strategies for resolving situations where no rule from the rule set applies to a case. One strategy resolves such conflicts by determining that no conclusion should be reached. The alternative strategy resolves that the most common class should be assigned to the case.

A number of rule refinement strategies may be applied to the rules developed through application of the DLG algorithm. The rules developed by the DLG algorithm usually refer to

all attributes from the domain. A number of these references will be logically redundant (stating that the attribute may assume any value) or in other ways spurious (placing unnecessary restrictions upon the rule.) Further, the DLG algorithm seeks to find the most specialized production rule set that correctly classifies all positive examples of a class and no counter-example. As a result, the specified ranges for some attributes are likely to be more restricted (limited to actual observed minimum and maximum values) than the ideal. The rule refinement options control rule refinement in an attempt to overcome these deficiencies.

Five rule refinement strategies are available, none, conservative, radical, fast conservative and fast radical. The first of these options, none, leaves the rules as developed by the DLG algorithm, except that clauses that require all possible values for a single attribute are deleted.

The remaining strategies are based on a linear search that sequentially deletes clauses and tests whether the resulting rule incorrectly covers any cases of another class. If so, the clause is reinstated. As the search is sequential (for the sake of computational efficiency) the order in which the clauses are examined can alter which clauses are deleted and which retained. Conservative rule refinement conducts two searches in different orderings. Clauses that are eliminated under both orderings are deleted from the final rule.

Radical rule refinement also conducts two searches in different linear orders. However, of the two resulting rules, it selects the one with the least clauses. Radical rule refinement then applies an algorithm that extends the bounds of ranges in the remaining clauses to positions in between that of the most extreme observed positive example and the closest observed counter-example.

Fast radical rule refinement conducts a single linear search and deletes all clauses that are redundant under that ordering. The bounds of ranges are also extended as in full radical rule refinement.

Fast conservative rule refinement is the same as fast radical rule refinement except that the bounds of ranges are not extended beyond the most extreme observed values from positive examples.

The user may constrain the system's actions during rule refinement by restricting the clauses that may be deleted. Associated with each class is a set of attributes called its key attributes. These are initialized to the empty set, and may be altered by the user. The clauses relating to the key attributes for a class may not be deleted during rule refinement.

Two user controlled options modify the search strategy used during the DLG induction process. The standard DLG algorithm is a single alternative non-backtracking search. One option allows this search to be applied multiple times, with the resulting characteristic descriptions being compared on lexical criteria and the most simple selected. This is the theory optimization search strategy, described in detail by Webb (1990.) The user may select any number of such searches to be employed.

The other search related option allows the user to select a rule optimization search strategy (Webb, 1990.) Under this option, multiple competing alternatives are maintained during the linear search. The best of these is selected at the end of the search on grounds of maximizing positive cover and lexical simplicity. The user can set the maximum number of alternatives that are to be maintained during the search.

ACQUIRING DOMAIN MODELS

One of the key aspects of the knowledge acquisition process is the development of the domain model. The current implementation of Einstein requires the user to specify the domain model

before any other activity takes place. The system would be greatly strengthened if it supported interactive domain model revision and extension (see, for example, Morik, 1989.)

In the context of the existing system, the domain model is defined by the attributes and their allowed values. Following their initial specification, it is not possible to add new attributes or values to those available.

It is intended that facilities for altering the domain model should be incorporated. These may be invoked by the user at any stage. They will also be invoked when the system encounters examples belonging to different classes which cannot be distinguished within the current model. Such a situation will trigger an interactive dialogue which will aid the user in extending the model. Extensions to the model may be expressed in terms of new attributes or new specifications of possible values for existing attributes.

Such an extension would enable the user to start with a very simple domain model and interactively refine it with support from the system as the production rule set evolved.

APPLICATIONS

Einstein is currently being operated by a number of users ranging from financial institutions that are using it for financial market analysis to a university department that is using it to develop a student course advisor.

The main evaluation of the system has occurred in the context of an investigation into techniques for non-surgical diagnosis of glomerular disease (Webb & Agar, 1991.) Currently, all patients suspected of significant glomerular disease undergo renal biopsy to obtain a specimen of the kidney, from which laboratory analysis derives a definitive diagnosis.

Attempts at statistical analysis of signs and symptoms that can be detected without surgical intervention have failed to develop non-surgical diagnostic techniques.

A preliminary pilot study has applied Einstein to an existing database containing such signs and symptoms along with diagnoses derived from laboratory analysis of biopsy specimens. As this database was not developed for this purpose, it does not contain many signs and symptoms that it is believed may be relevant to successful diagnosis. Further, the task is made even more difficult by the limited size of the database (284 cases) and large number of classes (31 distinct diseases.) Five of these diseases are represented by only one case each. Despite these formidable obstacles, an average diagnostic accuracy of 51% was obtained by 1000 expert systems, each derived from 80% of the data, randomly selected, and evaluated by application to the remaining 20%. This compares with a chance accuracy of just 3% for diagnosis by random selection between the 31 conditions.

Even more promising, when expert systems were created to diagnose specific diseases that were better represented in the data, far higher accuracies were obtained. Indeed, for the disease microscopic polyarteritis, of which the data contains seventeen cases, when one hundred expert systems were developed from different random selections of 80% of the data and each applied to the corresponding 20% remaining, every positive diagnosis was correct. That is, although some cases of microscopic polyarteritis were not correctly diagnosed, every case diagnosed as microscopic polyarteritis was indeed microscopic polyarteritis. If these results can be extended to a more comprehensive database, then it may be possible to derive diagnostic rules that, in cases where a positive diagnosis is reached, obviate the need for surgical biopsy.

The excellent results of the pilot study appear to be ample justification for extending the study to a larger, more comprehensive, database.

Several interesting results have emerged from experiments using Einstein's interactive facilities during this pilot study.

Many of the rules developed are initially regarded with skepticism by the domain expert involved. That is, they do not tally with his understanding of the domain. This initially led to doubt about the potential worth of the rules. This skepticism has been overcome, however, as a result of attempts, using Einstein's interactive facilities, to improve upon the rules derived by the induction system. Almost invariably, these lead to a decline in the diagnostic quality of the rules.

Further, the ability to evaluate the rules by interactive exploration of the positive and counter examples to each rule has demonstrably increased confidence in the quality of the output of the data driven machine learning sub-system.

This has led to an acceptance of the initially unpalatable proposition that it is to be expected that new knowledge should appear dubious and even challenging to those that are familiar only with the old.

CONCLUSION

Einstein has been aimed at two distinct user communities. The first of these are expert system developers that require fast development of expert systems in a context to which attribute value machine learning is appropriate. In such a context, Einstein can harness the respective strengths of both the induction system and the domain expert to provide rapid and sophisticated expert system development.

The second community at which Einstein is aimed is potential expert system developers with little or no previous expert system development experience. It is anticipated that the simple rule representation and editing mechanisms and the ability of the system to propose rules will provide a gentle introduction to the knowledge acquisition task for the novice.

It is intended that Einstein should provide a simple, yet sophisticated and supportive environment for expert system development. The facilities for rapid induction of initial rule sets and inductive revision of existing rule sets, coupled with the ability for the user to edit rules and interactively provide counter-examples, adds an additional tool to the knowledge acquisition tool-kit.

Einstein provides a unique combination of data-driven and model-driven rule development. The induction system provides data-driven input to the development process and the domain expert provides model driven direction to the data-driven component and model driven refinement of its output.

ACKNOWLEDGEMENTS

The Immunoglobulin A Nephropathy data set was compiled by Dr. John Agar of Geelong Hospital. His cooperation in making this data set available is greatly appreciated.

This research has been supported by the Australian Research Council.

BIBLIOGRAPHY

Bareiss, R. (1989) *Exemplar Based Knowledge Acquisition : A Unified Approach to Concept Learning.* Academic Press, San Diego.

Boose, J. H., Shema, D. B. & Bradshaw, J. M. (1989) Recent progress in Aquinas: A knowledge acquisition workbench. *Knowledge Acquisition*, **1**: 185-214.

Gaines, B. R. & Rappaport, A. T. (1989) A heterogeneous knowledge support system. *Proceedings of the Fourth Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff.

Michalski, R. S. (1984) A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds) *Machine Learning: An Artificial Intelligence Approach.* Springer-Verlag, Berlin, pp. 83-129

Morik, K. (1989) Sloppy modeling. In K. Morik, (Ed) *Knowledge Representation and Organization in Machine Learning.* Springer-Verlag, New York, pp. 107-134.

O'Niel, J. L. & Pearson, R. A. (1987) A development environment for inductive learning systems. *Proceedings of the First Australian Joint Artificial Intelligence Conference*, Sydney, pp. 134-145

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer & D. Mitchie (Eds.) *Machine Intelligence 5*, Edinburgh University Press, Edinburgh, pp. 153-163.

Plotkin, G. D. (1971). A further note on inductive generalization. In B. Meltzer & D. Mitchie (Eds.) *Machine Intelligence 6*, Edinburgh University Press, Edinburgh, pp. 101-124.

Quinlan, J. R. (1987) *Generating Production Rules from Decision Trees.* NSWIT School of Computing Sciences Technical Report 87.7.

Shapiro, A. D. (1987) *Structured Induction in Expert Systems.* Addison-Wesley, Reading, Mass.

Webb, G. I. (1990) Rule optimization and theory optimization: Heuristic search strategies for data-driven machine learning. *Proceedings of the First Japanese Knowledge Acquisition for Knowledge-Based Systems Workshop*, Tokyo, pp. 219-232.

Webb, G. I. (1991a) *Learning disjunctive characteristic descriptions by least generalisation.* Technical Report 2/91, Deakin University, Department of Computing and Mathematics, Geelong.

Webb, G. I. (1991b) Data driven inductive refinement of production rules. *Proceedings of the First Australian Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Pokolbin, pp. 44-52.

Webb, G. I. & Agar, J. W. M. (1991) The application of machine learning to the diagnosis of glomerular disease. *Proceedings of the 1991 IJCAI Workshop on Representing Knowledge in Medical Decision Support Systems*, Sydney.

Whitehall, B. L., Lu, S. C.-Y. & Stepp, R. E. (1990) CAQ: A machine learning tool for engineering. *Artificial Intelligence in Engineering*, **5**: 189-198.

APPENDIX A - THE FORMAT OF AN EXAMPLE FILE.

```
attributes
BX_NO : integer ignored
AGE : integer
SEX : {m, f}
POSTCODE : integer ignored
YEAR : integer ignored
DIAGNOSIS : decision { IGA_NX, negative }
SYST_BP : integer
DIAST_BP : integer
OEDEMA : {t, f}
HEADACHE : {t, f}
DYSPNOEA : {t, f}
H'PTYSIS : {t, f}
ARTHRALGIA : {t, f}
RASH : {t, f}
LOIN_PAIN : {t, f}
U.R.t.I.  : {t, f}
ANOREXIA : {t, f}
WT_LOSS : {t, f}
BOWEL_SX : {t, f}
U.RBC'S : integer
UTP : real
CRCL : real optional
S.CR : real
S.UREA : real 1 optional
TOT.PROT : integer optional
S.ALB : integer optional
IGG : real optional
IGA : real optional
IGM : real optional
C3 : real optional
C4 : real optional
ASOT : integer optional
S.CHOL : real 1 optional
S.TRIGLY : real 1 optional
S.GLUCOSE : real 1 optional
HB : real 1 optional
WCC : real 1 optional
PL'TS : integer optional
ESR : integer optional
ANF : {neg, pos} optional
ANTI-DNA : real optional
ANTI-GBM : {neg, pos} optional

examples
1     4     m     3232  1979  negative  105   65    t     f     f     f
      f     f     f     f     f     f     f     0     5     ?     40
      3.7   ?     ?     4.1   1.6   1.5   1.2   0.24  ?     6.5   3.4   ?
      14.8  8.1   317   ?     neg   ?     ?
2     4     m     3214  1979  negative  125   85    t     f     f     f
      f     f     f     f     f     f     f     12000       5     ?
      60    2.9   41    16    ?     ?     ?     1.8   0.39  ?     10.2
      6.3   ?     14.6  8.1   371   78    ?     ?     ?
```

A Framework for Integration of Machine Learning and Knowledge Acquisition Techniques

David C. Wilkins
Department of Computer Science
University of Illinois
405 North Mathews Ave
Urbana, IL 61801
wilkins@cs.uiuc.edu

Abstract

The fields of knowledge acquisition and machine learning share a major common goal, namely the creation and refinement of knowledge-based expert systems. Yet substantial barriers exist to substantial cross-fertilization between these two active research subfields.

The major purpose of this paper is the presentation of a framework to allow large-scale interaction and cooperation between knowledge acquisition and machine learning. The framework builds upon two contributions from expert systems and knowledge acquisition: generic role-limited shells and interactive acquisition tools that formalize the interface to the domain expert and knowledge engineer. Within this context, an interesting challenge for machine learning is to automate the knowledge transfer between interactive acquisition tools and the environment, knowledge base and performance element. This provides machine learning with an opportunity to investigate learning for complex methods of knowledge representation and inference. However, such an approach will require the creation of a national resource library of systems, tools and protocols.

1 Introduction

The fields of knowledge acquisition and machine learning have heretofore taken different approaches to the creation and refinement of knowledge-based expert systems. Researchers in knowledge acquisition have focused on and pioneered the development of interactive and semi-automated learning techniques (Boose and Gaines, 1988; Gaines and Boose, 1988; Boose, 1989; Brule and Blount, 1989; Boose and Gaines, 1990a; Boose and Gaines, 1990b; Scott et al., 1991). The research approach of allowing a human expert or knowledge engineering to be in the learning loop when creating an expert system has produced many important results. Our understanding of learning in rich task environments and for complex performance elements has been deepened (McDermott, 1988; Marcus and McDermott, 1989; Wilkins, 1990). And much of the task of creating a knowledge-based

expert system has progressed from being an unformalized activity between a knowledge engineer and a domain expert to being a formalized activity wherein this understanding is captured in interactive acquisition tools. Such tools can intelligently query domain experts and then refine and debug the knowledge base and performance element based on the domain knowledge they acquire (Boose and Bradshaw, 1988; Gruber, 1989; Musen, 1989; Porter et al., 1990).

The emphasis of machine learning researchers has been on scenarios that involve total automation of the learning task (Michalski et al., 1983; Michalski et al., 1986a; Kodratoff and Michalski, 1990). This has naturally restricted machine learning to simpler performance elements than those studied in knowledge acquisition. However, the benefits have been rich; there are now a broad range of different learning approaches to the classic problems like concept learning. For example, the solution methods for concept classification when given a library of pre-classified training instances now include a decision tree method (Quinlan, 1986), an AQ15 rule induction method (Michalski et al., 1986b), a backprop method (Hinton, 1989) an explanation-based method (DeJong and Mooney, 1986; Mitchell et al., 1986), a genetic algorithms method (Booker et al., 1989), a case based method (Lehnert, 1987), a classification regression analysis method (Breiman et al., 1984) and a statistical pattern recognition method (Fukunaga, 1972; Duda and Hart, 1973).

At the present time, greater interaction and cooperation between knowledge acquisition and machine learning should be advantageous for both fields. The purpose of this paper is to describe the substantial benefits to both fields from cross-fertilization, and to describe an approach to the achievement of large-scale cooperation. Toward this end, the paper is organized as follows. In Section 2, we review the relationship between performance element complexity and difficulty of learning of the performance element. This motivates Section 3 on generic role-limited shells, which are a class of performance elements that strike a good balance between complexity and learnability. Section 4 describes interactive semi-automatic acquisition tools for generic shells, which provide machine learning with a formalization of the learning problem and provide it with a new class of training examples. Section 5 shows that machine learning for expert systems can be nicely focused if it concentrates on providing answers to queries of the acquisition tools to human experts, knowledge engineers and the environment. In section 6, the approach is shown to very likely require the construction of a national resource library that provides to machine learning researchers with a suite of performance elements, knowledge acquisition tools, and acquisition protocols based on these tools. Lastly, Section 7, presents a summary and conclusions.

2 Effect of performance element complexity on learning

Generally, the more complex the knowledge structures of the performance element to be improved, the more difficult is the problem of automatic learning. Historically, the artificial intelligence community has intensively studied the learning of very simple and very complex performance elements, but little in between that is relevant to the creation of knowledge-intensive problem solvers. Concept classifiers are one of the simplest perfor-

mance elements in artificial intelligence, and learning the knowledge for these performance elements has been extensively studied within machine learning. Computer programs written in languages such as Lisp, Prolog and Pascal are very complex performance elements; the fields of automatic programming, intelligent tutoring and knowledge-based software engineering has extensively studied the learning of such programs from examples and specifications.

As an AI performance elements to be improved increase in complexity, the machine learning problem becomes in essence the problem of automatic programming. Hence, the lessons of automatic programming are of great importance to machine learning. This is especially the case because even though automatic programming has been a field of active research for a quarter of a century, we seldom turn to an automatic programming systems when we wish to create a Lisp or Pascal programs. Moreover, the size of the programs that can be automatically synthesized or debugged has not increased significantly in decades.

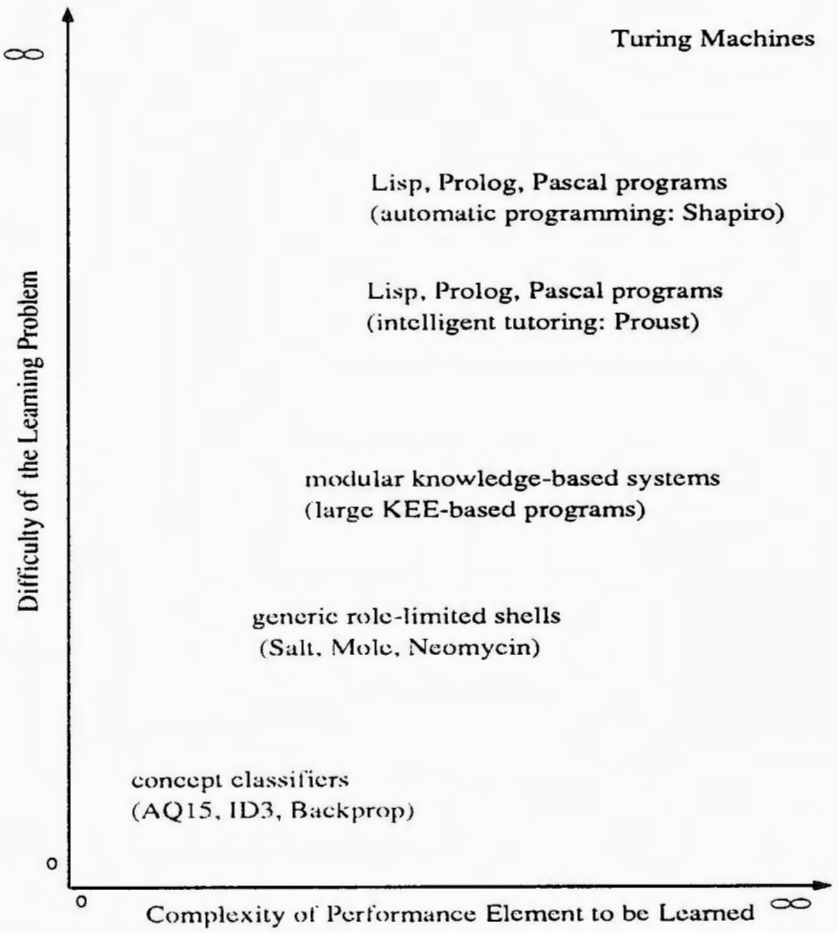


Figure 1: The difficulty of learning usually increases as the complexity of the performance element increases. This figure provides a very rough approximation of the typical difficulty of learning a typical member of a number of well-known program classes.

Automatic programming approaches include program synthesis from examples (Summers, 1977; Bauer, 1979) and specifications (Barstow, 1985; Smith et al., 1985). The challenges of automatic programming and the state-of-the-art can perhaps best be appreciated by work on the analysis or synthesis of sorting and searching programs (Harandi and Ning, 1990; Rich and Wills, 1990), equational programs (Dershowitz and Pinchover, 1990), and Prolog programs (Dershowitz, 1988; Shapiro, 1983).

Large knowledge-based expert system programs are now routinely developed, and we understand the broad range of representation, inference, and organization methods that are required by such programs (Buchanan and Smith, 1989; Waterman, 1988; Feigenbaum et al., 1989; Engelmore and Morgan, 1988). Currently, most of program creation, refinement, and debugging is performed manually; clearly an important goal for machine learning is to automated this process. However, the lessons of automatic programming should rightfully give us pause. There is no indication at present that machine learning techniques can succeed any better than automatic programming techniques. Hence, there is negative evidence with respect to having research in learning concept classifiers scale-up to the learning of the type of large and complex knowledge based systems that are routinely constructed today by manual methods.

Another perspective on the challenges of learning complex performance elements comes from the field of intelligent tutoring. There has been substantial work in trying to debug or find errors in student programs that are written in Lisp, Prolog, and Pascal. This is equivalent to the problem of trying to improve a student-generated program so as to meet some specification. Note that intelligent tutoring situations are much simpler to solve than the general automatic programming problem, because the tutoring program is able to make so many simplifying assumptions and draw on some very powerful sources of information.

One example in point is the Proust work for finding student bugs in relatively simple Pascal programs (Johnson and Soloway, 1985; Murray, 1988). The Proust work made the following simplifying assumptions: the problem was known beforehand, the program to be debugged was limited to about one-page of Pascal programs, and restricted to program was one of the first attempts of a novice programmer. And two powerful sources of information were available: that a complete and perfect program specification is available, and that empirical evidence is available on the types of errors that the students will make. Yet, finding the student errors was of limited success, and relaxing the restrictions to more realistic settings did not appear to be possible.

In the next section, we will describe a class of programs for which machine learning may be able to take an approach that will permit it to avoid the negative results of automatic programming.

3 Generic role-limited shells as performance elements

As artificial intelligence programs become more complex, the lesson from the field of automatic programming is that machine learning should not focus on the development of learning methods for first-order theories, general-purpose programming languages, or general-purpose production systems. Over many years the field of automatic programming has tried to do just this. However, a class of specialized languages called generic role-limited shells may avoid the difficulties encountered by the field of automatic programming.

The basic motivation for generic role-limited shells is the observation that many instances of expert-problem solving fall into a number of natural categories, such as interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction and control (Waterman, 1988). For each of these generic tasks, a shell can be constructed that encodes much of the strategy for control knowledge for solving many of the problems with a generic task class. By studying the method of the shell, the roles that each of the pieces of knowledge assumes during problem solving can be determined; this is why they are often referred to as role-limited shells (McDermott, 1988). These shells can serve as a basis for a knowledge acquisition systems. The Mycin experiments ushered in the idea of generic role-limited shells (Buchanan and Shortliffe, 1984; Davis, 1982). The most effective demonstration of their efficacy to semi-automating knowledge acquisition has been a large collection of systems developed by John McDermott and his group (Marcus, 1988; McDermott, 1988).

Knowledge-based expert systems based on generic role-limited shells deserve careful study for a number of reasons. To someone in artificial intelligence, they can be seen as being a major component in the construction of an intelligent autonomous agent, which is an ultimate goal of artificial intelligence. To a computer scientist, there are two reasons why this class of programs deserve careful study. First, they are the only non-trivial class of computer programs for which the art of computer program construction and debugging can probably be successfully mechanized in the next ten years. Second, it is likely that the first computer program that mechanizes this activity will itself be a knowledge-intensive expert system.

The use of a well-structured generic shell restricts the general problem of automatic program construction and debugging in four key ways. First, the computer program is written using an expert system shell for a specific generic problem class, such as heuristic classification and propose-and-revise design (Clancey, 1985; Marcus and McDermott, 1989). This facilitates debugging because the program code is highly modular and structured. Second, the strategy and control knowledge for the problem class can be explicitly represented in the shell (Hayes-Roth, 1985; Clancey, 1988; Park et al., 1989) This provides a conceptual framework that permits a learning and debugging program to more easily examine and reason about the program code. Third, deep domain models must be tied to the expert system using the predicate and function vocabulary of the shell (Karp and Wilkins, 1989). Fourth, the learning and debugging program must have knowledge about programming and knowledge about the specific application domain.

The use of generic role-limited shells has important consequences. By having the performance element be a role-limiting method, the structure of the knowledge base is determined. Not only is the type of knowledge known, but the roles it plays in problem-solving are known. The same benefits are seen when we look at the learning critic. By knowing what we want in the knowledge base, the knowledge engineer can be replaced by a elicitation tool that knows exactly what questions to ask of the user, or what templates to provide to the user to fill in. Moreover, because the performance element and the knowledge base have a rich yet known overall structure, the learning program can reason about the knowledge structures of these objects.

Many years from now, the field of machine learning may recognize the four major classes of programs shown Figure 3 as being special. These classes are integrated architectures (Soar, Prodigy), generic role-limiting shells (Salt, Oncocin, Neomycin), intelligent databases, and arbitrary general-purpose programming languages (Lisp, Prolog and Pascal).

4 Knowledge acquisition: A formalization of interfaces

This section describes a major results from knowledge acquisition which can assist machine learning research. The previous section described a major result from expert systems research that can assist machine learning research.

Much of what knowledge acquisition research has learned is captured in the interactive

	Special Purpose Programs	*General Purpose Programs*
Heuristic Problems	Generic Role-Limited Shells	Integrated Architectures
Algorithmic Problems	Intelligent Databases	Arbitrary Lisp, Pascal

Figure 2: The major classes of programs that machine learning might pursue. Generic role-limited methods are an interesting class of programs as a focus of machine learning research.

acquisition tools that have been created. These tools can be viewed as a formalization of the acquisition activity that occurred at one time as a natural language dialogue between the domain expert and knowledge engineer. Interactive tools like Salt have a vocabulary that can be understood by a domain expert.

A major contribution of knowledge acquisition is a formalization of the interfaces to the learning critic in Figure 3. There was a time when knowledge acquisition involved a dialogue over a cup of coffee. But now, the functions that the knowledge engineer performs has been formalized. That is, the knowledge engineer communicates with the environment and the knowledge base through formalized interfaces.

In the performance model of learning (Buchanan et al., 1978), the goal is to improve a problem solving program called the performance element. This is the mission of the learning critic. It is accomplished by refining and debugging a knowledge base, based on information from the environment.

Let us look at the formalized interfaces first. We can set as one goal of the field of automated knowledge acquisition to eliminate the need for the expert for one or more questions. That is, the automated knowledge acquisition program goes to the very sources used by the expert to get the knowledge, which might be textbooks, manuals, experience,

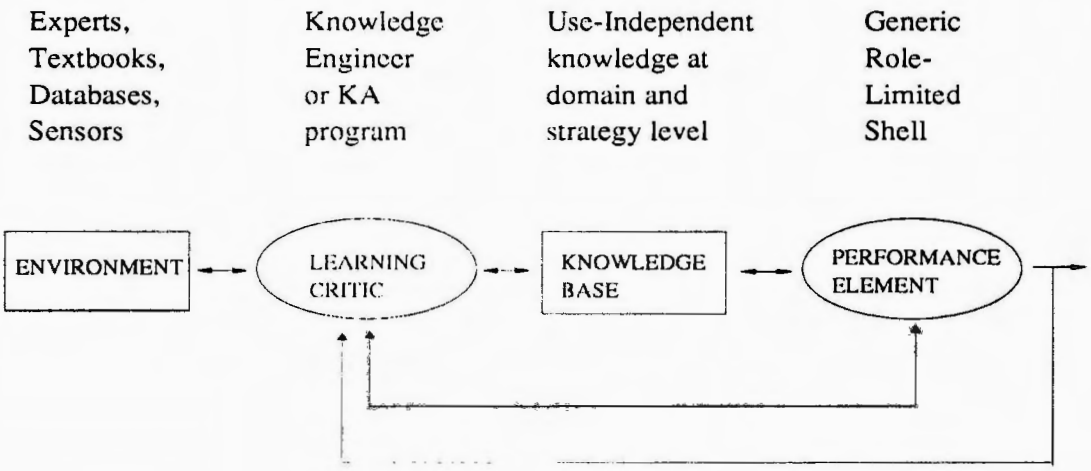


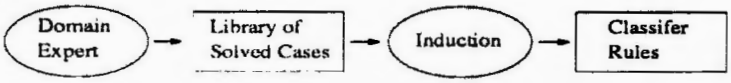
Figure 3: The well-known performance model of learning is shown (Buchanan78) and above the model is shown the way the model is specialized when doing automated knowledge acquisition for expert systems. In some knowledge acquisition systems such as Salt, the learning critic has been replaced by an acquisition program.

or just commonsense knowledge.

Some knowledge acquisition systems have effectively removed the knowledge engineer, such as Salt. By defining of a formal interface to the environment, there may be a possibility of machine learning techniques being used to eliminate or reduce the need for a domain expert.

Changing the modular code of a generic code is also a more structured and formal activity. Instead of dealing with staring over listings, there is an explanation facility by which the expert can communicate and gather knowledge about the program structures. Much of the perceptual element which plays such a major role in debugging programs such as Pascal has been eliminated.

Traditional Model of Machine Learning Research
for Knowledge-Based Expert Systems:



A Model of Machine Learning Research based on
Acquisition Tools and Generic Shells:

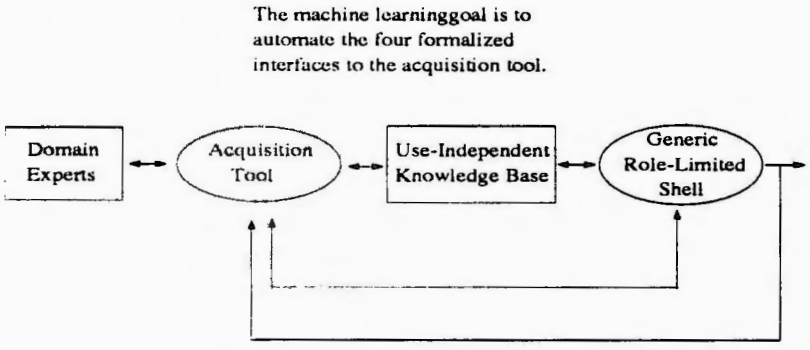


Figure 4: A comparison of current typical efforts to automate knowledge acquisition and an alternative approach

5 Machine Learning: Automating acquisition tool interactions

This section focuses on approaches to the use of machine learning techniques in a knowledge acquisition setting.

Consider relating learning to generic role-limited shells. The significance of our learning results could be expresses in terms of the method and or complexity of the knowledge representation and inference of the performance element. Or machine learning experiments could provide the basis for taking an existing performance element and reorganizing it so as to make the local credit assignment problem easier. In a similar vein, one can study a given representation with respect to observability and controllability.

Another main approach is to relate learning to an interactive knowledge acquisition tool, such as Teiresias, Protos, Ask, Salt. An advance would occur each time the machine learning system automates a piece of the interactive protocol that involves the domain expert providing assistance in in refining or debugging the knowledge base. This approach involves collect protocols of an elicitation tool interacting with the environment. Every time the tool extracts knowledge from an expert is a machine learning opportunity.

Another approach could begin with an examination of the expert problem solving to determine when the various strategies of learning enter into the problem solving, such as advice taking, induction, analogy, experimentation, etc. There are two needs: to make the learning critic aware of when it needs to use the different methods and to allow the learning critic to actually use the different methods.

This suggestions addresses the wide interest to find means to get various learning methods to work together within one framework. As is well known, the field of machine learning has investigated a large number of learning methods. Typical taxonomies of learning methods include rote learning, learning by taking advice, induction learning, explanation-based learning, deduction, analogy, experimentation, and discovery (Michalski et al., 1983; Dietterich and Michalski, 1983). For the most part, these methods have been investigated individually, although there have been exceptions (Lenat, 1976).

6 National Resource of Shells, Tools and Protocols

Almost every major talk on future research directions for machine learning emphasizes knowledge-based expert systems as a promising approach (Michalski et al., 1983; Michalski et al., 1986a; Kodratoff and Michalski, 1990). Yet there are very few machine learning researchers involved with expert systems.

To those of us who do research in this area, this should come as no surprise, because the entrance requirements are very steep. Let's consider some of the entrance requirements that must be paid *before* one can do research.

First, there is a need for a state-of-the-art second-generation expert system shells, such

as Salt (Marcus and McDermott, 1989), Minerva (Park et al., 1989) or BB1 (Hayes-Roth, 1985). One needs a state-of-the-art expert system shell, because it is the latest shells that provide the nicest structuring of the knowledge, which is where the leverage to learning appears to reside. But of the available shells there are very few that are robust, efficient, well-documented, and have well-structured code that can be extended for learning. That most machine learning researchers cannot even define a second-generation expert system shell does not help the problem.

Second, there is a need for one or more knowledge bases that work in conjunction with the shell code. Third, there is a need for interactive acquisition tools that can supply learning researchers with a new type of training instances for learning. Fourth, a good understanding of the knowledge acquisition process that is being automated often requires having a domain expert and a knowledge engineer. Fifth, it is well known that this types of learning research can benefit form first-principles theories of the domain, such as a large amount of background medical knowledge and library of cases in the domain of medicine.

As can be seen, no matter how inviting the line of research described in this paper, the resources that will be required by those wishing to combine machine learning and knowledge acquisition are orders of magnitude greater than those currently needed.

In sum, we see automated knowledge acquisition as requiring a state-of-the-art knowledge of three currently distinct subfields of artificial intelligence: expert systems, knowledge acquisition, and machine learning. And also resources and tools from these three areas. An analogy to language acquisition might be helpful. A learning method is very effected by our model or theory of natural language processing. This changes over time, and so language acquisition is faced with a moving target. Likewise with knowledge acquisition for expert systems. It is common to still see papers published that address the learning problem for first-generation Mycin-like systems.

One solution to allow large-scale research in automated knowledge acquisition is a national resource library, so that learning researches aren't burdened with construction of a state-of-the-art performance element, the need to develop manual tools that provide the basis of automation, and the need for a domain expert that actually uses the tools – as a prelude to doing research in automated knowledge acquisition. All of this can be available for distribution via ftp. There are many challenges to such an endeavor, such as the problem of proprietary software.

7 Summary and Conclusions

In this paper, we attempted to provide one research direction that points in the direction of doing learning for performance elements of of greater complexity that at present but of manageable complexity. Our approach centers around the notion of generic role-limited shells and formalized interfaces for programs that perform knowledge engineering. Because knowledge acquisition has has been willing to have one or more experts in this field has explored much more complex learning situation and performance tasks than ma-

chine learning proper. Hence there are advantages to machine learning building upon and contributing to the advances in knowledge acquisition.

8 Acknowledgements

This paper is a byproduct of the Automated Knowledge Acquisition track of the Ninth International Workshop on Machine Learning, which took place in July 1991 and attempted for the first time to bring together researchers in knowledge acquisition and machine learning. The paper benefited greatly from recent discussions with Ray Bareiss, Bruce Buchanan, Tom Dietterich, Brian Gaines, Tom Gruber, Mehdi Harandi, Ryszard Michalski, Tom Mitchell and Bruce Porter. This research was supported by ONR grant N00014-88K0124 and an Arnold O. Beckman research award from the University of Illinois.

References

Barstow, D. R. (November 1985). Domain-specific automatic programming. *IEEE transactions on software engineering*, 11(11):1321–1336.

Bauer, M. A. (1979). Programming by examples. *Artificial Intelligence*, 12(1):1–21.

Booker, L. B., Goldberg, D. E., and Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1-3):235–282.

Boose, J. H. (1989). A survey of knowledge acquisition tools and techniques. *Knowledge Acquisition*, 1(1):3–37.

Boose, J. H. and Bradshaw, J. M. (1988). Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. In Boose, J. and Gaines, B., editors, *Knowledge Acquisition Tools for Expert Systems*, volume 2, pages 39–64. Academic Press.

Boose, J. H. and Gaines, B. R. (1988). *Knowledge Acquisition Tools for Expert Systems.* Academic Press.

Boose, J. H. and Gaines, B. R. (1990a). *The Foundations of Knowledge Acquisition.* Academic Press.

Boose, J. H. and Gaines, B. R. (November 4-9, 1990b). *Proceedings of the Fifth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop.* Calgary, Canada: SRDG publications.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees.* Monterrey, CA: Wadsworth Press.

Brule, J. F. and Blount, A. (1989). *Knowledge Acquisition.* McGraw-Hill.

Buchanan, B. G., Mitchell, T. M., Smith, R. G., and Johnson, C. R. (1978). Models of learning systems. In Belzer, J., editor, *Encyclopedia of Computer Science and Technology*, chapter 11. New York: Marcel Dekker. Also Stanford Report STAN-CS-79-692.

Buchanan, B. G. and Shortliffe, E. H. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, Mass.: Addison-Wesley.

Buchanan, B. G. and Smith, R. G. (1989). Fundametals of expert systems. In Barr, A., Cohen, P. R., and Feigenbaum, E. A., editors, *The Handbook of Artificial Intelligence, Volume IV*, chapter XVIII, pages 149–192. Addison Wesley, New York.

Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27:289–350.

Clancey, W. J. (1988). Acquiring, representing, and evaluating a competence model of diagnosis. In M. Chi, R. G. and Farr, M., editors, *The Nature of Expertise, 1986*, pages 343–418. LEA Press.

Davis, R. (1982). Application of meta level knowledge in the construction, maintenance and use of large knowledge bases. In Davis, R. and Lenat, D. B., editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 229–490. New York: McGraw-Hill.

DeJong, G. and Mooney, R. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–177.

Dershowitz, N. and Pinchover, E. (1990). Inductive synthesis of equational programs. In *Proceedings of the 1990 National Conference on Artificial Intelligence*, pages 234–239, Detroit, MI.

Dershowitz, N. and Lee, Y. (1988). Logical debugging. Technical report, University of Illinois.

Dietterich, T. G. and Michalski, R. S. (1983). A comparative review of selected methods for learning from examples. In Michalski, Carbonell, and Mitchell, editors, *ML 1*, chapter 3, pages 41–81. Tioga, California.

Duda, R. and Hart, P. (1973). *Pattern Classification and Scene Analysis.* Wiley, New York.

Engelmore, R. and Morgan, T. (1988). *Blackboard Systems.* Addison-Wesley.

Feigenbaum, E. A., McCorduck, P., and Nii, H. P. (1989). *The rise of the expert company.* New York Times Books.

Fukunaga, K. (1972). *Introduction to Statistical Pattern Recognition.* Academic Press, New York.

Gaines, B. R. and Boose, J. H. (1988). *Knowledge Acquisition for Knowledge-Based Systems.* Academic Press.

Gruber, T. G. (1989). Automated knowledge aquisition for strategic knowledge. *Machine Learning*, 4:293–336.

Harandi, M. T. and Ning, J. Q. (1990). Knowledge-based program analysis. *IEEE Software*, 7(1):74–81.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26(3):251–321.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence*, 40(1-3):185–234.

Johnson, W. L. and Soloway, E. (1985). PROUST: Knowledge-based program understanding. *IEEE Trans. Software Engineering*, SE-11(3):267–274.

Karp, P. D. and Wilkins, D. C. (1989). An analysis of the distinction between deep and shallow expert systems. *International Journal of Expert Systems*, 2(1):1–37.

Kodratoff, Y. and Michalski, R. (1990). *Machine Learning: An artificial intelligence approach*, volume III. Morgan Kaufmann, San Mateo.

Lehnert, W. G. (1987). Case-based problem solving with a large knowledge base of learned cases. In *Proceedings of the 1987 National Conference on Artificial Intelligence*, pages 301–306, Seattle, WA.

Lenat, D. B. (1976). *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search.* PhD thesis, Stanford University.

Marcus, S., editor (1988). *Automating Knowledge Acquisition for Knowledge Based Systems.* Kluwer Academic Publsihers, Boston.

Marcus, S. and McDermott, J. (1989). SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–37.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Knowledge Based Systems*, chapter 8, pages 120–146. Boston: Kluwer Academic Publsihers.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1983). *Machine Learning: An Artificial Intelligence Approach.* Palo Alto: Tioga Press.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors (1986a). *Machine Learning: An Artificial Intelligence Approach*, volume II. Los Altos: Morgan Kaufmann.

Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N. (1986b). The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 1986 National Conference on Artificial Intelligence*, pages 1041–1045, Philadelphia, PA.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80.

Murray, W. R. (1988). *Automatic Program Debugging for Intelligent Tutoring Systems.* Morgan Kaufmann.

Musen, M. A. (1989). Automated support for building and extending expert models. *Machine Learning*, 4(3-4):347–376.

Park, Y. T., Tan, K. W., and Wilkins, D. C. (1989). MINERVA: A knowledge based system shell with declarative representation and flexible control. Technical Report UIUC-KBS-89-001, Department of Computer Science, University of Illinois, Urbana-Champaign, Illinois. Revision 1, May 1990.

Porter, B. W., Bareiss, R., and Hlote, R. C. (1990). Concept learning and heuristic classification iin weak-theory domains. *Artificial Intelligence*, 45(3):229–263.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Rich, C. and Wills, L. M. (1990). Recognizing a program's design: A graph-parsing approach. *IEEE Software*, 7(1):82–89.

Scott, A. C., Clayton, J. E., and Gibson, E. (1991). *A Practical Guide to Knowledge Acquisition.* Addison-Wesley.

Shapiro, E. H. (1983). *Algorithmic Program Debugging.* Cambridge: MIT Press.

Smith, R. G., Winston, H. A., Mitchell, T. M., and Buchanan, B. G. (1985). Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of the 1985 IJCAI*, pages 673–680, Los Angeles, CA.

Summers, P. (1977). A methodology for LISP program construction from examples. *Journal of the ACM*, 24.

Waterman, D. (1988). *A Guide to Expert Systems.* Addison-Wesley, Reading, MA.

Wilkins, D. C. (1990). Knowledge base refinement as improving an incomplete and incorrect domain theory. In Kodratoff, Y. and Michalski, R. S., editor, *Machine Learning: An Artificial Intelligence Approach, Volume III*, pages 493–514. San Mateo: Morgan Kaufmann.

A Comparison of Structured vs. "Unstructured" Interviews for Knowledge Acquisition

Larry E. Wood, Thomas C. Davis, Samuel L. Clay and John M. Ford
Department of Psychology
Brigham Young University
Provo, UT 84602-1130
e-mail WoodL@yvax.byu.edu

ABSTRACT

We have developed a structured interview technique designed to elicit domain concepts and objects and their organization in the first stages of knowledge elicitation, particularly the initial interview. We are conducting empirical studies to evaluate its effectiveness. This study compares our technique to a condition which encourages experts to describe their problem solving knowledge in whatever manner is most comfortable to them. We are just beginning to analyze the data and will have results to report at the workshop.

INTRODUCTION

Over the last several years we have developed a set of structured interviewing techniques, which we have used successfully on knowledge elicitation projects (e.g., Wood and Ford, in press; Ford and Wood, in press; Wood, Hansen, and Ford, under review). We have begun a series of empirical investigations to evaluate the effectiveness of these techniques relative to other interviewing methods, and this report describes our progress thus far. Our techniques draw on the disciplines of anthropology and ethnography (e.g., Spradley, 1979; Werner and Schoepfle, 1987) and cognitive science (e.g., Anderson, 1983; Ericsson and Simon, 1984; Kuipers, Moskowitz and Kassirer, 1988). First, our approach relies on the distinction between declarative and procedural knowledge as discussed by Anderson (1983). Declarative knowledge includes the conceptual entities (both concrete and abstract) in a particular domain and their various relationships. Procedural knowledge, on the other hand, is comprised of the decisions and procedures required for solving problems using relevant domain concepts and relationships. Another way to describe this distinction is a knowledge of "what" vs. a knowledge of "how".

Our approach to knowledge elicitation consists of four phases with guidelines for questions suited to each phase. First, the Descriptive Elicitation phase is intended to reveal the important entities and objects in the domain as reflected in the terms and specialized language used by the expert. A second phase, Structured Expansion, is designed to probe the relationships between the domain objects and the organization of the expert's knowledge using the terminology uncovered in the previous stage. The Scripting phase relies on the declarative knowledge found through the two previous stages to discover procedural knowledge, and the final component, Validation, is important throughout the process of knowledge elicitation to ensure that the knowledge being elicited is correct and adequate to enable a system in which it is implemented to solve the class problems with which it is concerned.

In this investigation, we have combined aspects of the first two phases in an attempt to assess their utility as a strategy for an initial interview. In Descriptive Elicitation, a variety of questions are designed to encourage the expert to talk about the domain at a general level, using domain-specific terms, which reflect the concepts and objects used in solving domain problems. Examples of such questions are "Could you describe the major steps you go through in doing X", or "Could you describe a typical case where you do X." The concepts and objects mentioned by the expert in answering Descriptive Elicitation questions are then used by the knowledge engineer in asking Structured Expansion questions such as "You mentioned that your performance depends on the type of X you encounter. Could you describe the different types of X for me?", or "You

mentioned you use either X or Y in performing the task. Are they both the same type of thing?" The knowledge engineer continues to question the expert in this manner until a relatively complete description of relevant domain objects is obtained, organized by their hierarchical relationships (e.g., types, kinds, parts, attributes, features, examples).

In discussing the "neglected first phase", Cooke (1989) has made a case for the importance of eliciting domain objects early in knowledge elicitation to make use of a variety of knowledge elicitation techniques. It is a well-known fact that experts' knowledge is highly organized and integrated, compared to that of novices (e.g., Chi, Feltovich, and Glaser). It is this organization that we attempt to uncover early on with our techniques. We have found in our own work in knowledge elicitation, for example, that it is much easier to interpret experts' detailed descriptions of procedural knowledge in a domain (e.g., a think-aloud problem solving protocol), once a relatively thorough understanding has been obtained of the concepts and objects used by the experts in problem solving.

In addition to assessing our interviewing techniques, we have an interest in learning if there are preferred strategies of explanation used by experts when they are given little or no structure. Other researchers (e.g., Gordon and Gill, 1989) have suggested that experts prefer a depth-first strategy, where they begin with a particular part of the process and explain it in detail before revealing other more general aspects. On the other hand, there is reason to believe that it is more helpful for a knowledge engineer, who is often a relative novice in the domain of interest, to have experts proceed in a breadth-first manner, where they give a broad overview of the entire process, then fill in details a "level" at a time (Zeitz and Spoehr, 1989).

In order to explore "typical" explanation patterns among experts and to provide a useful baseline against which to compare our interview techniques, we attempted to create a situation that allowed experts maximum freedom in describing their problem solving knowledge. This was accomplished by using a technique similar to Rogers' (1964) non-directive interviewing strategies. After asking the experts to describe how they went about accomplishing a particular task, the interviewer responded by reflecting back some of what was said and then inviting the experts to continue describing their problem-solving processes in more detail. Because we provided minimal prompting to the experts for their explanations, we call this condition minimal prompting (MP), and our structured techniques as structured prompting (SP).

In summary, the purpose of our initial investigation was two-fold. First, we needed a baseline situation with which to compare the effectiveness and efficiency of our techniques at eliciting domain objects and concepts and their organization. Second, we were interested in determining if there were consistent patterns in the strategies experts use when asked to describe how they go about solving a particular class of domain problems, but when they are given minimal direction in doing so.

METHOD

A Domain of Expertise

As discussed by (Burton, Shadbolt, Rugg, and Hedgecock, 1990) one of the difficulties in doing research involving expert-level problem solving is getting sufficient access to a suitable pool of experts. The domain chosen for the present investigation was personal clothing construction (sewing). A faculty member from the department of Clothing and Textiles[1] and the director of the custom sewing shop operated by that department assisted in the selection of a suitable sub-domain of knowledge suitable for the investigation. It consisted of a set of women's blouses consisting of all combinations of the characteristics shown in Figure 1 and to be constructed from either plain, 100% cotton oxford fabric or plain organdy. These two individuals then acted as experts in

providing a description of the knowledge required to construct the blouses. Their description served as a standard against which our experimental interviews could be compared.

Participants

The participants were students who were either employed in the custom sewing shop or other students referred by them, initially judged to possess a similar level of skill. Participation was solicited either in person (by showing each potential participant Figure 1) or via a telephone conversation (by listing the characteristics of the blouses shown in Figure 1). In both cases the potential participants were asked if they felt qualified to describe how to make any of those blouses or blouses with any combination of their characteristics, without needing to refer to the detailed instructions (guide sheet). Those willing to participate were paid $5.00 for engaging in an interview lasting approximately one hour.

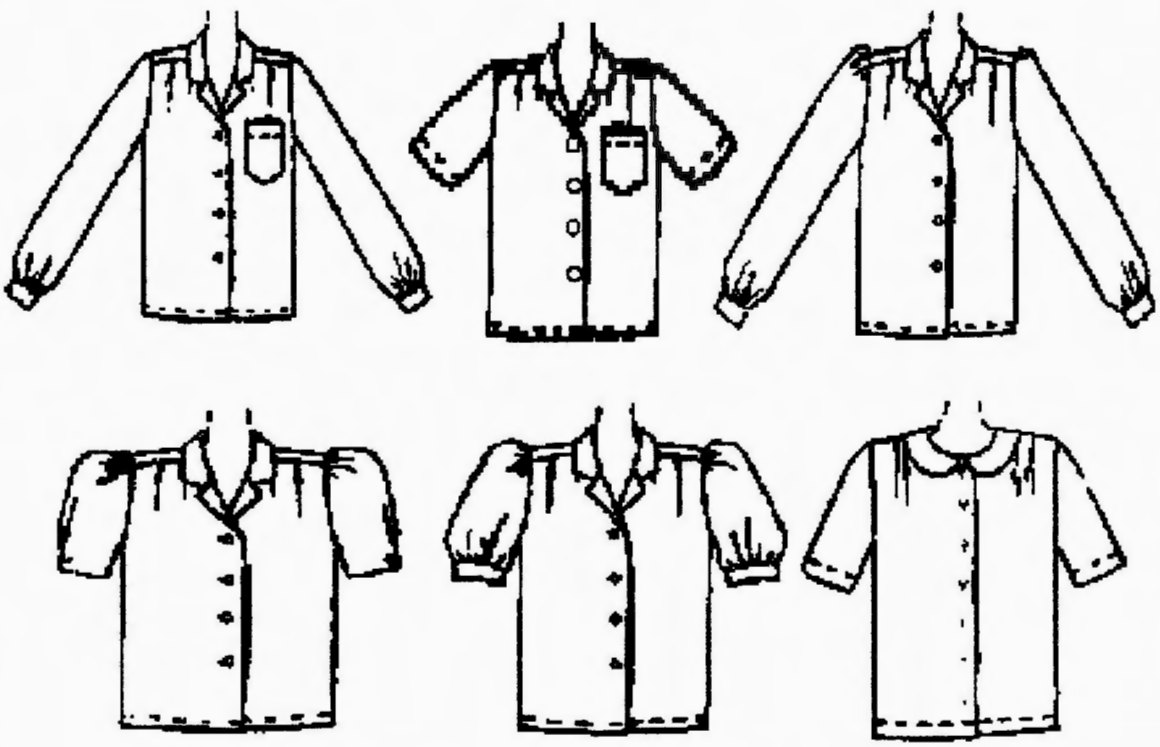


Figure 1. Blouse patterns used in the study

Following the interview, the participants were asked to respond to a questionnaire, asking the length of time they had been involved in sewing and the frequency of their having constructed various types of blouses and other garments. The results of that survey showed the fourteen participants as a group had been making clothing items for a mean of 9.6 years, had made a mean of 29.1 items during the past year, had made a mean of 171.4 total items, and had a mean of 71.1 experiences in explaining to others (teaching or assisting) how to make items of clothing.

Procedure

One of the inherent characteristics of a structured interview is that the expert is often reminded of what has already been said each time the interviewer asks a question (e.g., "You mentioned there are different types of sleeves. Could you describe each of them and how they are different").

Thus, it seemed important to provide some kind of reminding for the MP condition in order for the comparison to be valid. Also, research in memory retrieval (Watkins and Tulving, 1984) is very definitive about the effectiveness of contextual cues. To provide appropriate context, we gave the participants access to Figure 1, the pattern pieces used in constructing the blouses depicted there, and a sample of the two fabrics (cotton oxford and organdy) mentioned earlier.

The participants were randomly assigned to one of the two interview conditions in a counterbalanced order as they were scheduled for the interviews. The interviewers were the second and third authors, each of whom interviewed an equal number of participants in each condition. The interviews were conducted in the following manner. Both conditions began with the interviewer showing Figure 1 to the participant and saying, "During the next hour I would like for you to tell me as much as you can about how you go about making these blouses or a blouse from any combination of the characteristics shown here, either from 100% cotton oxford or organdy fabric. Assume that all I know about clothes is that I wear them. Notice that we have available the pattern pieces and a sample of cotton oxford and one of organdy, all of which you may use in any way you would like during the interview. Please take a moment to look over these items and tell me when you are ready."

When the participants indicated their readiness, those in the SP condition were asked, "Could you describe the general steps you would go through to construct any of the blouses pictured here, or those constructed from combinations of them?". This is an example of a task-related, grand tour question (Wood and Ford, in press), designed to encourage an expert to provide domain-specific information, but at a general level. Those in the MP condition were asked to begin to describe how they would make the blouses, doing so in whatever way they felt most comfortable. In both conditions, the participants were then allowed to talk until they indicated they were finished.

When those in the SP condition finished, they were asked a Structured Expansion question (Wood and Ford, in press) about one of the terms mentioned in their previous statements (e.g., "You mentioned there are different types of sleeves. Could you describe each of them and how they are different"). For the remainder of the interview, the interviewer continued to ask Structured Expansion questions, attempting to elicit both the concepts and objects in the domain and their relationships. When those in the MP condition finished talking, they were told by the interviewer, "I am going to review some of the main points you mentioned, and I would like you to pick one of them and provide more details about it." The interviewer continued in this fashion for the duration of the interview.

RESULTS

Each of the interviews was tape-recorded and transcribed. At present, we are just beginning our analysis of the transcripts. Our intent is first to analyze them for the concepts and objects and their relationships by constructing hierarchical concept maps such as that shown in Figure 2. This will be accomplished using a knowledge editor we have been developing called SNOWMAN (Wood and Ford, 1991). The two conditions will then be compared on the total number of concepts and a number of measures we are developing to reflect the degree of organization among the objects. Because our method is designed to elicit the organization of the objects as well as the objects themselves, we expect a higher degree of organization to result in the SP condition compared to the MP condition. One of our initial impressions from conducting the interviews is that our experts are not very comfortable with the lack of direction in the MP condition. There is a tendency for them to pick one of the blouses and begin to describe how to make it, specifically. This is consistent with the observations of Gordon and Gill (1989) that experts prefer a depth-first strategy for explanation.

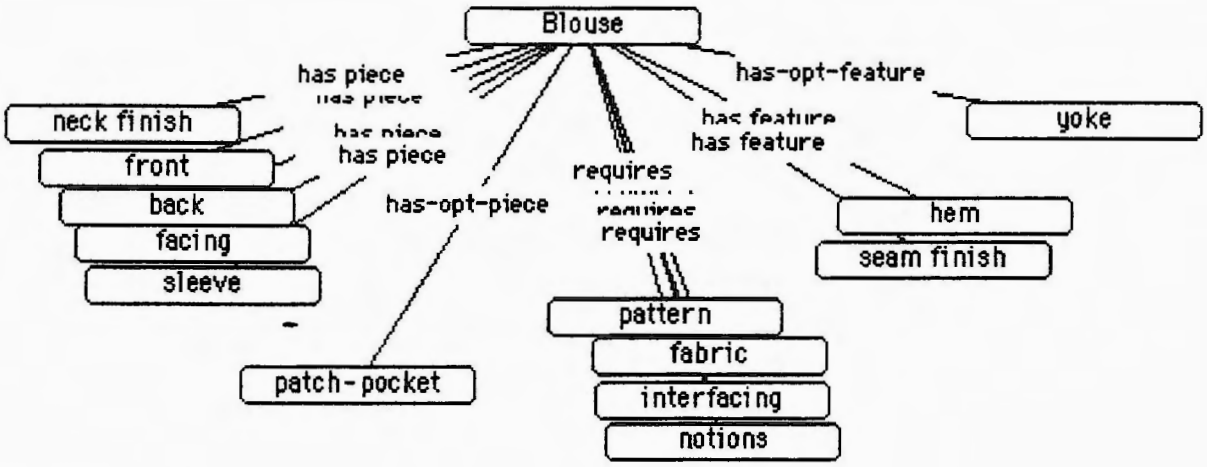


Figure 2. An Hierarchical Concept Map

---

[1]We express our appreciation to Sue Lamerson and Della Steinekert for their assistance in selecting the domain and providing a description of the relevant knowledge for solving the problems.

REFERENCES

Anderson, J. R. (1983). *The Architecture of Cognition.* Cambridge, MA: Harvard University Press.

Burton, A. M., Shadbolt, N. R., Rugg, G., and Hedgecock, A. P. (1990). The efficacy of knowledge elicitation techniques: a comparison across domains and levels of expertise. *Knowledge Acquisition,* 2, 167-178.

Chi, M.T.H., Feltovich, P.J., and Glaser, R. (1981).Categorization and representation of physics problems by experts and novices," *Cognitive Science*, 5, 121-152.

Cooke, N. J. (1989). The elicitation of domain-related ideas: Stage one of the knowledge acquisition process. In C. Ellis (Ed.)*Expert knowledge and explanation: The knowledge-language interface* (pp. 58-75). New York: Wiley and Sons.

Ericsson, K. A., and Simon, H. A. (1984). *Protocol analysis: Verbal reports as data.* Cambridge, MA: MIT Press.

Ford, J.M. and Wood, L. E. (in press) Structuring and Documenting Interactions with Subject-Matter Experts. *Performance Improvement Quarterly.*

Gordon, S. E. and Gill, R. T. (1989). Question probes: A structured method for eliciting declarative knowledge. *AI Applications in Natural Resource Management*, 3(2), 13-20.

Kuipers, B., Moskowitz, A. J., and Kassirer, J. P. (1988). Critical decisions under uncertainty: Representation and structure. *Cognitive Science*, 12, 177-210.

Rogers, C. R. (1964). Toward a science of the person. In T.W. Wann (Ed.), *Behaviorism and Phenomenology.* Chicago: The University of Chicago Press.

Spradley, J. P. (1979). *The ethnographic interview.* New York: Holt, Rinehart and Winston.

Watkins, M. J., and Tulving, E. (1975) Episodic memory: When recognition fails. *Journal of Experimental Psychology: General*, 109, 251-278.

Werner, O. and Schoepfle, G. M. (Eds.). (1987). *Systematic Fieldwork: Vol. 1. Foundations of ethnography and interviewing.* Newbury Park, CA: Sage Publications

Wood, L. E., and Ford, J. M. (in press) Structuring interviews with experts during knowledge elicitation. *International Journal of Intelligent Systems: Special Issue on Knowledge Acquisition.*

Wood, L.E. and Ford, J.M. (1991). Techniques and Tools to Support the Knowledge Engineer in Structured Interviewing. Presented at the Fifth Annual European Knowledge Acquisition Workshop, Crieff, Scotland, May 21.

Zeitz, C. M. and Spoehr, K. T. (1989). Knowledge organization and the acquisition of procedural expertise. *Applied Cognitive Psychology*, **3**, 313-336.

Developing K-ONCOCIN:
A Case Study in the Cognitive Process of Knowledge Engineers

J. Brian Woodward,
Knowledge Science Institute
Department of Computer Science
University of Calgary
2500 University Dr. NW.
Calgary, Alberta
CANADA, T2N 1N4

woodward@cpsc.ucalgary.ca

Abstract

The domain of knowledge engineering reflects a strong emphasis on the acquisition and modelling of knowledge structures and the processes to manipulate and transform them. Knowledge engineering activity has been largely focused on the epistemological and representational aspects of knowledge. A different, yet complementary view would identify knowledge engineering as the interaction of cogniting agents with the emphasis on the cognitive processes used in knowledge engineering. This paper presents the results of an intensive discussion between the author and Marc Linster who acted as the interviewee. The topic of discussion focused on his activity with Mark Musen during the modelling stages of knowledge engineering activity in the development of K-ONCOCIN which is a KADS conceptual model of a part of the ONCOCIN system. For each stage of this 'bottom-up' modelling, the tasks were identified, the available knowledge and information resources identified, the classes of cognitive processes inferred, and the outcome structures described. This paper is best read in conjunction with Linster and Musen (1991).

1. Knowledge Engineering as a Set of Cognitive Processes

1.1 Principles

Knowledge engineering activity has been largely focused on the epistemological and representational aspects of knowledge. A different, yet complementary view would identify knowledge engineering as the interaction of cogniting agents with the emphasis on the cognitive processes used in knowledge engineering. The current approach in knowledge engineering forces us to view the outcome or the product of our efforts as most important. The 'knowledge' is 'acquired', 'transferred', 'captured' and 'modelled'. This emphasis directs our attention away from the processes that 'produce', 'organize', and 'represent' the knowledge: away from notions of learning, comprehending, and communicating. Perhaps these processes are taken for granted; perhaps they are considered invariable. This knowledge-intensive emphasis is clearly displayed in our tools, techniques and procedures in that they focus on structures of knowledge and the processes used to manipulate them. Few of our tools address the complexity of the cognitive processes used to develop the knowledge structures.

Wetter and Woodward (1990) have identified the importance of incorporating an understanding of cognitive processes in the development of theories of knowledge acquisition. In order to develop a method of knowledge acquisition in a principled manner, the epistemic concepts must be clearly defined, the epistemic concepts must have a basis in

psychological occurrence, and the representational formalism must reflect the intended epistemic and psychological meaning. This section sets out the premises for viewing knowledge engineering as a set of cognitive processes rather than as a set of activities which acquires and models knowledge structures in one form or another (Woodward et al, 1991):

1. Knowledge engineering represents an ordered collection of cognitive activities.

2. The purpose of knowledge engineering is to use cognitive processes to produce a model. Viewing knowledge engineering as a set of cognitive processes means that the emphasis is placed on the information processing activities which lead to a final model rather than on the characteristics and content of the final model.

3. Knowledge engineering is done in a cognitive environment. The environment defines a set of situations in which information is generated and produced, provided, manipulated, organized, re-organized and re-represented. The environment also defines those processes and structures which are used to generate, manipulate, comprehend, and organize the information. These processes are concerned with meaning, not just information. Consequently, the focus is on the cognitive processes of developing meaning.

4. The cognitive environment is characterized by sites and levels of cognitive activity. The environmental structure determines the locations of activity and the types of cognitive activity which occur at those locations. 'Sites' refer to those clearly distinguishable points of cognitive activity (eg. the knowledge engineer, expert). 'Level' refers to various combination of sites which are in interaction. Interaction of cognitive processing sites entails another level of cognitive activity which is distinguishable from cognitive activity occurring at a single site.

5. An understanding of the cognitive environment acts as a basis for developing tools, techniques, and procedures to support and/or replace these processes. With our emphasis on the cognitive aspects of knowledge engineering, the focus of potential support processes is based on our understanding and careful elucidation of those cognitive processes which are characteristic of sites and levels of cognitive activity. Rather than emphasize knowledge support we are suggesting a complementary view of cognition support as a basis for tool, technique, and procedure development.

1.2 Sites and Levels of Cognitive Activity

Figure 1. schematically identifies the cognitive environment as proposed. The diagram depicts five distinct sites of activity which intersect in the knowledge engineering context (the box). The domain context (A) is that site of cognitive activity that generates the problems and situations which require attention by the other systems. The knowledge engineering context (B) refers to the cognitive activity which is used by knowledge engineers while accomplishing their goals. The expert (C) site reflects those cognitive processes used to solve problems from site A. The user (D) site contains those cognitive processes used to solve problems from site A with the help of site E. This last site is that of the final knowledge-based system and reflects cognition-like properties. Initially the system at site E is non-existent but it begins to build from the outset and acts as a focal point for discussion and interaction.

Activities of interest at each site are those cognitive processes (and their supports) which are used to create meaning within the site. Here the emphasis is on cognitive processes

which are affected or influenced by few, if any, interactions with other sites. The cognitive processes here are considered solitary from the point of view of the other sites. Although information may be obtained from other sites, the cognitive activity of interest here are those processes which reflect how information is processed to create meaning. For example, the cognitive processes used by a knowledge engineer to create useful content categories from materials from the domain are identified at this site. The same could be said for both the expert and the user sites. Once the system site is developed, the cognition-like activity at this site is of interest from a cognitive environment perspective. Site B, the knowledge engineer, is the focus of activity for this case study.

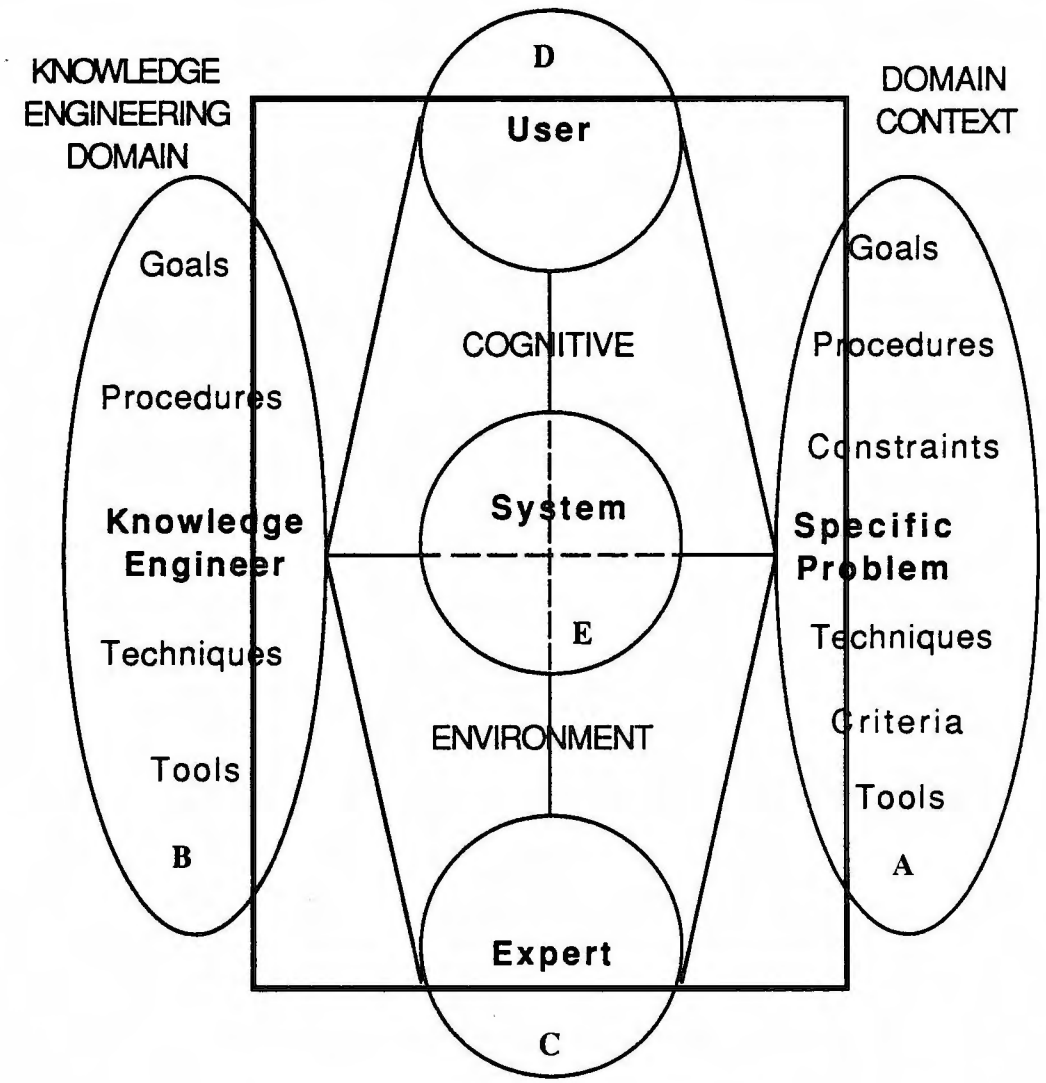


Figure 1. Sites of Cognitive Activity

2. Design of the Case Study

2.1 Introduction

Woodward et al (1991) have identified the importance of considering the cognitive processes used during knowledge engineering. The knowledge engineer specifically, becomes a site of important cognitive activity which is necessary for the development of the knowledge-based system. The initial purpose for addressing the knowledge engineer's cognitive processes was to identify tools, techniques, and procedures which may be developed as cognition aids during knowledge engineering activity. Another purpose emerged after the initial data collection was completed. It was decided that an important step would be to link (loosely at present) knowledge engineering tasks with the cognitive processes used to complete the tasks. This linking would help to further elucidate specific tasks, block them into stages, and then to help identify cognitive processes (including biases) likely required for task comprehension and completion.Stages of knowledge engineering activity could then be characterized by distinct cognitive processes and various forms of cognitive bias. Once these characteristics were determined, a number of tools, techniques, and procedures may then be developed as cognitive aids to knowledge engineering and decision-making.

The choice of knowledge engineer for this study was Marc Linster and the activity of interest was his and MarK Musen's re-modelling of ONCOCIN into a KADS model to produce K-ONCOCIN (Linster and Musen, 1991). This paper describes the authors' testing of the KADS modelling methods on a large, complex, medical domain task (cancer-chemotherapy administration). The choice of Marc Linster was made on the basis of availability and because the knowledge engineering activity included modelling in a restricted, well-defined domain.

2.2 Interview Process

Information was gathered from a single, three-hour, interview between the author and Marc Linster. The interview focused on the sequence of activities completed by Marc Linster in the role of knowledge engineer in developing the KADS model of ONCOCIN (K-ONCOCIN). The interview was semi-structured in that different types of questions were prepared beforehand but the asking of the questions was influenced by the content and flow of discussion. The goal was to identify and describe the steps taken, the decisions made, the information used and required, the outcomes, choice points, and the degree of confidence of various decisions. At each well-defined step in the process a number of questions were asked (for example):

How did you define or perceive the current task?
What were the steps in the task?
What information did you have to begin with?
What information did you use and how did you use it?
What information did you need but not have?
What decisions did you make at this point?
How did you make this decision?
What other options were available?
What did you use to help you make your decision?
How did you know you made the best decision?
What questions or unresolved issues did you have after the decision?

Notes were taken during the interview, later condensed and transcribed, and returned to Marc Linster for verification. On the basis of this interview and the paper an (Linster and Musen, 1991) analysis procedure was established and executed.

2.3 Analysis of Results

The purpose of the analysis was to identify the stages of knowledge engineering activity from the beginning point of initial knowledge structures to a final formal KADS representation. At each stage the knowledge or information sources were determined, likely classes of cognitive activity were postulated, and the outcome knowledge structures were described. Table 1. outlines four stages of knowledge engineering activity. Each is characterized by task demands, available knowledge resources, cognitive processes and biases, and the outcome structures. The cognitive processes are described as classes because no attempt was made to formally test for the existence of certain processes. These processes were inferred by the interviewer. It was assumed that many more cognitive processes may have been operating so those identified in the table should not be considered as complete nor as entirely mutually exclusive. Also, it was assumed that the classes were not mutually exclusive. The 'inference' class of cognitive processes refers to those processes used during text comprehension and include semantic and schematic inferences commonly used when reading textual material in a natural language format. The 'complexity reduction' class refers to those cognitive processes identified as pattern recognition, abstraction, and include a range of 'reductive' biases. The 'judgment and choice' class refers to processes of comparison, construction, and labelling patterns.

The next three sections provide a detailed discussion of the case example accompanied by selected research results which support the results of the analysis.

3. Stage One: Orientation to the Domain

3.1 Description of the Task

The knowledge engineering activity began with a 'well-formed' domain structure represented by highly ordered information and complex knowledge structures. However, the structures were not initially understandable to the knowledge engineer due to the complexity of the domain content (cancer-chemotherapy administration). As well-structured as the domain appeared, the knowledge engineer was faced with the task of quickly understanding the structure before he could attempt any knowledge engineering activity to re-model the domain.

This initial step for the knowledge engineer is cognitively demanding in that he is expected to enter into the epistemological structure of a domain far enough to build a model of that domain. Even if the domain is restricted in scope, the knowledge engineer must become familiar enough with the problem-solving structures in the domain and with the underlying assumptions of the domain of interest to build a model of the domain. These demands are similar to expert development within a domain in that the knowledge engineer, as a novice, must quickly become familiar with the necessary domain concepts and problem-solving processes in order to adequately model the domain or part thereof. When a knowledge engineer addresses new, complex, epistemological structures, he/she uses a variety of cognitive processes to first understand and comprehend the basic structures and assumptions of the domain and second, to model this understanding.

Stage of Knowledge Engineering	Task Demands	Available Knowledge/Inform. Resources	Class of Cognitive Processes	Cognitive Biasing Processes	Stage Outcomes
Initial Orientation to the Domain	•develop basic understanding of domain structures -address new epistemological structures -identify underlying structures	•treatment protocols •ONCOCIN 'dumps' •documents	•text comprehension -semantic and schematic inferences	• 'prior analogy' •'common connotation' •comprehension failure	•identification of comprehension failure -faulty inferences •strategy shift
Development of Initial Domain Structure (Domain Model)	•reduce complexity of the domain -reduce domain structures -make connections among structures	•expert •sequential nature of the protocols •meanings of the frame slots	•pattern recognition -identify patterns -match patternss -construct patterns •pattern anticipation -look for patterns	•judgmental biases -cue familiarity, clarity, availability •selective perception •reductive biases -prior analogy -step-wise bias	•understanding of frame slots lead to concept hierarchies • sequential nature of protocols lead to activity graphs •principle 1
Development of Inference Structure (Inference Model)	•label observed, recurring patterns using KADS methodology	•anticipated patterns in data protocols (activity graphs) •informal domain-independent inference terms •KADS library	•categorization •class assignment •abstraction •selection •dealing with uncertainty	•heuristics •cue availability, clarity and frequency •competing alternatives	•basic inference model in personal terms •list of labelled classes for roles and activities •principle 2
Development of KADS Representation (Conceptual Model)	•model the informal inference model into the KADS representation system -creat a new KADS model	•KADS library and methodology •intermediate inference model -labelled patterns	•comparison of abstraction •pattern construction •pattern decomposition •category assignment	(similar to last stage)	•new KADS interpretation model •prototype for K-ONCOCIN •principle 3 • KADS conceptual model

Table 1. Stages of Knowledge Engineering for K-ONCOCIN Development

3.2 Available Information/Knowledge Structures

The initial set of knowledge structures came in three formats. The first was a set of treatment protocols which consisted of a sequence of tests, possible X-ray therapies, and possible chemotherapies developed by the North California Oncology Group (NCOG). The protocols also contained all the permissible sequences of tests and treatments for any patient (see Figure 1., Linster and Musen, 1991). The protocols were written in a 'reduced-text' form and were highly compiled and stylized. The knowledge engineer noted that "the protocol documents were difficult to understand for someone outside the medical profession". The knowledge engineer, however, needed these protocols to provide the structure for the modelling process but found "no direct and obvious way to build a model from the protocols [because they did]... not explicate the knowledge in such a way that could be used directly in a modelling process".

The second format for knowledge structures consisted of approximately 34 pages of ONCOCIN 'dumps' which were lists of frame-like objects used by ONCOCIN. Each of these structures contained a label and numerous named slots. The knowledge engineer noted that these representations were decontextualized knowledge units which obtained their semantics from the ONCOCIN inference engine. The knowledge engineer was able to make little sense of the protocols and data structures together.

The knowledge engineer also had a series of documents which were intended to help with understanding the frames and their slots. These documents consisted of clinical explanations of the protocols and knowledge structures written for a knowledgeable population familiar with the domain. The knowledge engineer found that the documents contained too many unfamiliar concepts to be of use to him at this point in the process. He characterized his initial steps in the modelling process as having to make sense of the "interaction between formalized data structures representing bits of knowledge out of context and textual information..[written].. for a clinical audience with a different purpose in mind".

3.3 Cognitive Processes

This initial stage of knowledge engineering placed a number of cognitive demands on the knowledge engineer. The cognitive processes involved at this stage largely reflect the inference processes based on prior world knowledge, plausibility, and pragmatics. These natural language, text-based inferences constitute a class of cognitive processes used in text comprehension to understand concepts, to differentiate between concepts, and to make connections between and among concepts. As someone reads, he/she attempts to make the text meaningful, usually by making predictions (Smith, 1982) and then by determining if the predictions hold true. The elimination of possible meanings until the correct one is found is a process of inferencing based on prior knowledge and on the goal of the reader (van Dijk and Kintsch, 1983; Lesgold and Perfetti, 1978). These types of processes are of interest to knowledge engineers because of the onerous inferential demands placed on the knowledge engineer in order to make meaningful the information contained in domain textual material,.

It is very common for readers to fail in making all the necessary inferences during text comprehension (Britton et al, 1990). Comprehension failure increases as the text becomes less of a narrative (with common inference patterns) and more expository. It is common for a knowledge engineer to read expository and or instructional texts. Making the appropriate inferences depends in large part on prior knowledge of the domain and it is clear that considerable inferential demands are placed on the knowledge engineer when he/she encounters a new domain. The more inferences required of the reader by the text, the less

likely the inferences will be made and the less likely the intended meaning will be constructed. Making the necessary distinctions of concepts and making the necessary inferences of these concepts requires familiarity with the domain: something which the knowledge engineer does not initially have and may never develop sufficiently during the knowledge engineering activity of the project.

The difficulties of the knowledge engineer are further defined by Garner (1985) who notes that the comprehension of technical, expository prose is "little assisted by content-schematic knowledge, in that the very novelty of the material ensures that the readers will be unable to fit much of the new information into their old in-head information"(p.10).

At this stage of understanding and due to the cognitive demands placed on the knowledge engineer, a number of potential biasing processes are also likely. Due to the complexity of the domain, 'prior analogy' and 'common connotation' biasing processes (Feltovich et al, 1989) were possible. These two processes refer to how complexity is reduced in new, confusing domains. The former occurs when a new concept is given meaning based on already held concepts within the domain or from other domains. The latter occurs when technical terms are reduced in meaning based on the day-to-day use of the term or concept.

Another potential biasing process may occur at the meta-cognitive level. This type of bias involves comprehension failure and the detection of this failure (see Baker and Brown, 1984a, 1984b; and Brown et al, 1986 for reviews). Often readers do not pay attention to all the words of a sentence or passage. Also, sentences are often mis-parsed due to a partial match between text-based terms and the relevant knowledge structures (Reder and Cleermans, 1990; and see also a summary of error detection research, Garner, 1988). Finally, the necessary inferences may not be made by the reader. Under these circumstances, comprehension will be incomplete and the reader may fail to note the discrepencies. Comprehension failure can only be rectified or corrected if the reader becomes aware of the errors or aware of the partial nature of the inferences. If detection fails as well, then few connections or inferences are likely to be made. Markman (1977) has called this state of affairs the 'delusion of comprehension'.

3.4 Outcomes

The knowledge engineer recognized very quickly that he was unable to comprehend the initial knowledge structures to the extent necessary for model building. The protocols gave some indication of the treatment process, the frame structures gave very detailed information but without context, and the text materials provided a great deal of information but of too great a complexity. Essentially, the knowledge engineer experienced difficulty in generating the necessary inferences required, based on the available materials, to 'make sense' of the domain. However, he identified this situation and attempted to solve some of the difficulty by embarking on attempts at structuring the domain. These attempts are outlined in the next sections.

4. Stage Two: Development of Initial Structure

4.1 Description of the Task

The goal in this stage was to discover a pattern of relationships among the domain concepts provided. The purpose was to reduce the complexity of the domain to a simpler state for successful knowledge engineering. Domain layer analysis was described by the knowledge engineer as providing "insight into the interrelations and dependencies among domain concepts which, in turn, allows the knowledge engineer to perceive recurring inference

patterns". The knowledge engineer's initial attempts at structuring the domain were based on the knowledge structures available to him. After he experienced difficulty in basic understanding of the domain, he embarked on a variety of structuring methods of which he was familiar. He needed to apply his own tools, techniques and procedures to the domain. At this point he chose a strategy of determining if the frame data-structures would 'map' onto the protocols in some repeatable manner. He was hoping to develop a set of rules for mapping one to the other and to use the documentation to help him understand the meanings of the slots in the frame data-structures. However, there was little success in this activity. Although the knowledge engineer was motivated to find patterns in the protocol/frame structures, he could generate little confidence in his results. It was clear at this point finding a direct mapping between data structures and the medical protocols was not possible.

4.2 Available Knowledge/Information Resources

Now, the knowledge engineer needed his expert. He discussed his situation with Mark Musen and was able to gain a much better understanding of two essential characteristics of the domain. The first was a much better grasp of the meaning of the slots in the frame structures. The second was a deeper understanding of the 'sequential' nature of the activities outlined in the treatment protocols (see sections 2.2.1 and 2.2.2, Linster and Musen ,1991). Based on this deeper understanding, the knowledge engineer was able to renew his efforts at finding patterns and rules for relating the two types of knowledge structures. This type of activity reflected a 'bottom-up' approach to modelling because the KADS methodology did not offer any guidance of what patterns to look for at the domain level.

4.3 Cognitive Processes

Based on the new information the knowledge engineer obtained about the slots in the frame structures, he was able to use some common knowledge engineering structures and processes to organize the frames and their content. He developed concept hierarchies to hold and order the various configurations of frame types (Figure 8, Linster and Musen, 1991). These initial structures of concept definitions were developed to reflect the frame hierarchies in OPAL (Musen,1987). These structures "contributed to the terminology for the analysis of the transition networks that OPAL provides ONCOCIN". These structures provided the basis for the domain layer in KADS.

Based on the characteristic of sequential activity outlined by the protocols, the knowledge engineer shifted from a direct mapping approach to the use of a visual flowchart-like structure he called an 'activity graph'. This type of graph was used to display the various types of activities and their sequences. In order to develop an initial inference structure, the knowledge engineer used one protocol (NCOG 2083-1) and used the ONCOCIN data-structures which were used to phrase the ONCOCIN equivalent of the protocol in order to build the activity graph (see Figure 9. Linster and Musen, 1991). He used the frame structures as the symbols on the graph and labelled them using domain terms. Now the knowledge engineer was able to read the protocol and understand it by thinking about it in the frame language. He was able to see how the activity graph gave him the needed perspective to match the data-structures with the protocol.

The predominant class of cognitive activity in this stage is characterized as 'pattern recognition'. The reason for activating this type of process is to reduce dramatically the complexity of the domain. In developing the concept hierarchies, a process of 'identify and match' was likely used. The construction of concept hierarchies required the identification of frames and, based on the meaning of their slots, the frames were assigned or matched to

the node positions in the concept hierarchy in a way that the meaning of the frame is maintained in conjunction with the neighbouring nodes in the hierarchy.

By using the activity graph, the knowledge engineer was forced to look for regularities in the protocols which were sequential in nature. This process could be called ' identify and order' and was based on elaborating the regularities he found. The knowledge engineer began to find patterns he described as consisting of:

1. an activity or sequence of activities, terminating in
2. a test or end point, which lead to
3. a resulting set of other activities or sequences of activities.

He saw these structures as 'cycling' or repeating throughout the data set.

Once these patterns were found, another cognitive procedure became apparent. The knowledge engineer began to 'anticipate' similar patterns. He began to look for 'cycling' patterns of activity and began to construct a number of these. Also, he began to identify more complex patterns which he constructed using the more simple ones. Complex patterns occurred more often when the protocol called for combined chemo treatments or when treatments were used in parallel (eg. chemo and radiation). When the knowledge engineer defined these more complex patterns as structured combinations of the more basic patterns, they were not the same structures as indicated by the expert.

Margolis (1987) has provided insight into some of the cognitive activities used in pattern recognition tasks. He uses these cognitive processes for explaining different forms of judgment and decision-making. Judgment is defined as more than an ability to gradually construct new routines (learning by trial and error) and more than the capability to choose among regular and standing routines in a particular situation (choice). Judgment is defined as the ability to construct new responses to a challenge, to solve a problem or difficulty by a process of inventiveness. Patterns of variables, the juxtaposition of groupings and sub-groupings of variables, and patterns of consequences are identified until one appears correct for the context. Margolis postulates four cognitive processes which contribute to this activity. Something about a perceived pattern of variables may appear wrong or ineligible. This prompts a 'checking' or 'look-again' response for this pattern of variables. Then a 'lumping' process occurs which brings another pattern forward. With repeated checkings, a pattern or combination of patterns begins to emerge which may then 'inhibit' other patterns from being considered or 'prime' other patterns to be considered.

In these kinds of tasks a number of biasing processes have been identified. The first type is that of 'judgmental' biases (Hogarth and Makridakis, 1981) and the second is that of 'reductive' biases (Feltovich et al, 1989). Difference in cue clarity, availability and frequency can lead to different judgments. The more frequently certain cues arise indicating patterns or potential patterns, the less likely other, less frequent, cues may be recognized. Likewise, the clearer and/or more perceptually available the cue, the more likely it will be selected and the less likely cues will be perceived with lower clarity and availability. The result is a potential loss of useful information or the loss of potentially important but seldom identified patterns.

Biases due to framing lead to selective perception. In the case considered in this paper, the knowledge engineer anticipated seeing certain kinds of patterns in the activity graph. This provided a 'frame' for his perception. The question arises as to what extent patterns were not identified because they did not fit the frame of reference. This type of bias may or may not have occurred; however, the knowledge engineer noted that he did not test his understanding of the patterns and cycles on other than one protocol. He did not analyse

another protocol directly into the frame/activity graph form to test whether his frame of reference was generalizable.

Any attempt to identify patterns in data results in reducing the information available. Reductive biases have been identified in the biomedical domain (Feltovich et al, 1989,1984; Spiro et al, 1988, 1990). These researchers address the nature of conceptual understanding in biomedicine. More specifically, they address how deep models of complex ideas develop and how problems may arise with these models, during their development, which lead to misconceptions. The results have shown developmental patterns for a variety of biomedical misunderstandings. The results support three main reasons for misunderstandings: multiplicity, interdependency and oversimplification. The first refers to the fact that many influences contributed to the acquisition and maintenance of misconceptions such as learner style and ability, the methods of education, and practices of biomedical science research. The second characterizes misconceptions as reciprocating networks based on basic component ideas which are faulty. These networks tend to support each other or positively reinforce each other so that misconceptions are promulgated throughout the network. The final type suggests that due to human information processing capacity, the complexity of the concepts is reduced by selecting, collapsing, or relating concepts in a way which changes the meaning.

Complex concepts make unusual cognitive demands on the novice. Feltovich et al (1989) describe four main demands on:

1. working memory because of the large number of nested or looped steps or goals, or because of the large number of variables to be considered, processed, and/or reconciled;

2. formal representation in the sense that the degree of abstraction necessary for understanding is often onerous (eg. representing the concept of rate or acceleration);

3. intuition or prior knowledge in the sense that new concepts can conflict with prior meanings and/or relationships among concepts; and

4. notions of regularity because many concepts can be ill-structured and highly variable in how they are used or they may be very strongly dependent on other concepts for their meaning.

These same demands are placed on the knowledge engineer. These cognitive demands affect knowledge engineering activity by leading to misconceptions which may be perpetuated into the final system if not 'caught' by the expert (assuming there exist specific procedures for the expert to use to 'catch' the misconception). Also, when complex conceptual domains are addressed, complexity reduction occurs. Many researchers have studied the processes whereby humans use particular procedures to reduce the complexity of conceptual information. (Kahneman and Tversky, 1973; Kahneman et al, 1982; Meyer and Booker, 1991; Cleaves, 1986; Coulson et al, 1986; Spiro, 1980).

Feltovich et al (1989) distinguishes conceptual bias from those biases in judgment and decision-making (Kahneman et al, 1982) in that the family of 'reductive biases' are used solely to reduce complexity rather than reflect judgment processes. Six reductive biases were identified. 'Static' bias occurs when a dynamic system is viewed as a static model. 'Step-wise' bias describes the situation where a continuous process is broken down into discrete, identifiable steps resulting in a loss of meaning. 'External agent' bias is the attribution of essential intrinsic characteristics of entities or processes to external influences. 'Prior analogy' bias occurs when new concepts are given meaning based on already held

concepts or simple models within the domain or from other domains. 'Common connotation' bias occurs when technical terms are reduced in meaning based on the day-to-day use of the term or concept. Finally, 'restriction of scope' bias refers to the belief that general principles are taken to apply only under certain circumstances based on a repeated co-incident relationship.

4.4 Outcomes

The most useful outcomes were the concept hierarchies and the activity graphs. The first provided the basis for accurate terminological classification and the second a basis for an initial inference model of the domain. Both of these knowledge structures was essential for building a KADS inference model.

The graphs provided a means to identify patterns of activity. Without these it was impossible to link in any meaningful way the data-structures with the protocols. Although no test was applied to determine if the use of the activity graphs was appropriate or the 'best choice', the graphing activity was extremely useful for the knowledge engineer in developing a set of rules for identifying and consistently placing the frames and their terms in the recurring patterns. From this activity, the knowledge engineer identified a principle for later work: *all patterns should reflect the most basic form, they should be repeatable, and complex patterns should be built from simpler, more basic, ones.*

5. Stage Three: Development of the Interpretation Model

5.1 Task Description

In this stage of activity, the knowledge engineering task was to take the observed, recurring patterns and label them in a way which conformed to the KADS requirements. More abstract descriptors of the identified patterns were required. At this stage the knowledge engineer's purpose was to develop or 'abstract-out' of the activity graph identifiable and describable generic patterns. The KADS approach suggests that this identification and labelling activity be completed in domain-independent terms to make them reusable: that patterned, domain-layer phenomena be described in domain-independent terms which then act as the components of the inference model. The knowledge engineer described this activity as a "process of interpreting raw data and thus is an iterative process that involves considerable uncertainty".

5.2 Available Knowledge/Information Resources

The knowledge engineer had available the set of activity graphs with domain terms and with recurring patterns identified and anticipated. He also had available to him the KADS interpretation model library and his own informal set of domain-independent category system for inference processes.

5.3 Cognitive Processes

From the previous stage, the knowledge engineer identified recurring patterns from the activity graphs. Now, his task was to classify these patterns into categories based on similarities and differences which he determined.

Processes of making distinctions are concerned with how categories are constructed and with how objects are assigned to categories. These processes are the basis for grouping objects and for the development of taxonomies. Categories have three main characteristics

(Smith, 1990): categories are used to code experiences to reduce the amount of stored detail; hierarchically arranged categories provide taxonomies of objects; and categories are used for inductive inference processes. Also, existing categories and their qualities are used to generate new categories. Once an object is categorized, this allows inferences to be drawn from and to the category (Gelman and Markman, 1986).

The point of interest for knowledge engineering is how such categories are distinguished, described, and learned. Estes (1989) discusses two main approaches to human category learning: storage an retrieval models and models based on abstraction from prototypes. Estes finds support for both models from a variety of research areas. Research in selective perception addresses the question of how an individual processes a great deal of information. The question arises as to what is saved and what is discarded. Another question addresses whether the information is first encoded with later selection or whether information is filtered out early on. Research on semantic memory addresses similar problems but in different terms. Storage of information in semantic networks suggests that information which specifies the meaning of a concept is stored at an appropriate node in a network. The concept is accessed when the meaning is required for judgments. Attributes and other relations are also stored. An alternate view suggests that information is stored in feature lists and that relationships are computed between words at the time of judgments. Category research also addresses whether a prototype is developed early on in information processing and relationships computed when needed and/or whether all information is stored in exemplar memory and consulted when a judgment is required. Although both models have been tested (Smith and Medin, 1981; Estes, 1986) each shows some support under certain conditions.

Once the patterns were distinguished and categorized based on salient features, the next task was to label the categories in such a way as to retain meaning while following as close as possible the recommendations of the KADS modelling approach. At this point the knowledge engineer did not use the library of models and inference terms suggested by the KADS method. His initial attempts to map the 'action' terms or verbs from his classes of actions failed because he was not able to 'make the leap' from the domain layer terms in the activity graph patterns to the domain independent terms suggested by the KADS inference library. The terms the knowledge engineer chose to describe these patterns were from his own personal set of terms based on his experience and on his reading of the 'problem solving' literature.

Once again, the bottom-up approach to modelling was necessary. The knowledge engineer used a collection of personal terms to understand and label the sequences of actions. Generally, verbs were labeled as 'activities' and nouns were called 'roles' or 'intermediate results' (see Wielinga, et al, 1991 for a similar alteraction in KADS terminology). The knowledge engineer mapped domain terms onto his inference structure by identifying the type of action indicated by the verbs and then selected or constructed a term to describe the class of activities which this terms suited. This same process was completed for the two classes of nouns.

Essentially, the KADS inference library offered domain-independent inference terms which where too abstract for use by the knowledge engineer. He was unable to select any terms which he was confident would retain the meaning of the domain layer terms. So, he constructed an intermediate layer of terms (a model) which reflected a strong flavour of the domain but was abstract enough to allow significant reduction in the complexity of the domain (see Table 3, Linster and Musen, 1991). The cognitive processes here are generally those of 'abstraction' through generating categories and selecting labels. A sub-process may be described as 'class instantiation' in the sense that the knowledge engineer had to make a number of decisions as to whether a particular pattern was an instance of a class.

This latter point brings to the surface the potential biasing processes in this stage of knowledge engineering. The knowledge engineer used his own 'heuristics' to make many of the judgments. He noted that he did not test whether his categories or labels were 'correct' or whether the terms he chose or constructed were the 'best' ones to use. Another problem here for the knowledge engineer was maintaining a consistent use of his terms. He noted that a useful tool would be one which provided him with a library of how he had used the particular activity or role terms in the past. Because this task was so uncertain, the knowledge engineer noted that his process of identification and labelling of patterns was only one of many ways to view the domain and inference layers.

The task was characterised by uncertainty which likely affected the knowledge engineer's judgment processes. These interaction effects between task demands and human judgment processes help explain some of the difficulties individuals have with making judgments under certain circumstances. Cue availability constitutes one contextual variable, reflecting aspects of task demand (Hogarth, 1987), which strongly influences predictive and evaluative judgments. Availability of appropriate cues (Tversky and Kahneman, 1973) can affect judgments of relative frequency and information presentation patterns can affect the salience of context and cues. Often the degree of variability of a variable is strongly distorted by contextual effects, thus affecting resulting judgments. For example, people tend to base their estimates of frequencies and probabilities on absolute vs relative frequencies (Estes, 1976). For this study, the relative frequency of cues may have lead to the construction of greater or fewer categories.

Within the knowledge engineering domain it is possible to identify and outline those judgment processes which are not normally completed effectively by the knowledge engineer. Statistical models have been developed or adapted (Meyer and Booker, 1989) but other types are possible. Hogarth (1987) has described different methods for combining information to stabilize judgment processes. Processes of judgment and choice are viewed as conflict resolution models in that the selection of competing alternatives requires a resolution of conflicting information. 'Compensatory' models directly address the conflict created by the differences by allowing trade-offs on the choice dimensions. Examples are the linear model, additive difference model, and the ideal point model. Each model sets a tactical approach to combining information. 'Non-compensatory' approaches are those which do not allow trade-offs on choice dimensions. Examples of these types of models are the conjunctive, disjunctive, lexicographic, and elimination-by-aspects models.

Limited human information processing capacity requires that we reduce the complexity of information by a number of methods, which consequently result in loss of information and meaning. The domain displays its own inherent patterns. The knowledge engineer and the expert (and the user) place their own patterns on the domain to make evaluative and predictive judgments and to deal with their own uncertainty about the perceived patterns displayed by the domain. Hogarth (1987) has identified a number of sources of error and bias in judgment in a general population. Meyer and Booker (1989) and Cleaves (1986) have studied the effects of bias with respect to expert judgment elicitation in the knowledge engineering domain and in the field of risk analysis. They have identified a number of cognitive processes involved in complexity reduction by the expert. Little work, however, has been done concerning the judgment bias of the knowledge engineer and the user.

As the knowledge engineer engages in complex, increasingly uncertain domains, the greater the need to develop aids to judgment. Although many procedural tools have been developed (Keeney, 1982; and von Winterfeldt and Edwards, 1986) to aid the decision maker, few appear applicable to the knowledge engineering domain.

5.4 Outcomes

The outcomes from this stage reflect the abstraction and selection processes of the knowledge engineer. The main knowledge outcome was the list of labelled classes of roles and actions which helped reduce the complexity of the domain in a principled manner. The knowledge engineer developed another principle which stated that whereas *it is necessary to map domain terms into a restricted language, this needs to be done in a manner which retains domain 'meaning'*. This is done to retain a focus for dialogue with he expert and to ensure the integrity of the inference model.

6. Stage Four: Development of a KADS Representation

6.1 Task Description

The goal in this stage was to map the domain and inference structures into a KADS representation and to complete the development of a KADS conceptual model. This type of model in KADS helps the knowledge engineer by providing a domain-independent framework to guide analysis, interpretation, and model formation of the domain. The problem-solving method of the domain is represented in the interpretation model and is used to identify the knowledge types required for problem solving. A library of interpretation models aids in the appropriate selection of a model either by direct selection and application or by modification of an existing model to 'fit' the problem. When a model does not exist for the domain at hand, the knowledge engineer must generate or construct a new one. Linster and Musen (1991) note that "...this [constructing a new model] happens quite frequently". The final conceptual model contains the interpretation model and the domain knowledge.

6.2 Available Knowledge/Information Resources

The knowledge engineer had identified classes of activities, roles, and intermediate results and had labelled sub-categories of these. However, he knew enough about the KADS library of interpretation models to know that he was not faced with an 'analysis' type problem and that the only 'synthesis' type interpretation models were too general. He knew also that no KADS model represented the repetitive or cyclic nature of the patterns he developed from the activity graphs. His own abstractions of the patterns were the basis for knowing that he needed to create a new interpretation model.

6.3 Cognitive Processes

The knowledge engineer found that he could not make a mapping from his inference terms to an existing KADS interpretation model so he needed to construct one. The first cognitive process of this stage used to reach the conclusion that an existing KADS model was not appropriate entailed a 'comparison of abstractions'. The knowledge engineer needed to develop the dimensions for comparison and then had to actually complete a certain number of comparisons. This type of cognitive process requires the knowledge engineer to use consistently a set of rules for comparison, for determining if one concept or structure means the same as another or has a similar enough meaning to fall into the same category. The need for this comparison is so similar that inferences can be made about the two concepts or structures so that meaning is maintained. The categorization research outlined in the last stage applies here as well.

The second set or class of cognitive processes addresses the 'construction' of a new KADS interpretation model. The construction process outlined in the Linster and Musen (1991)

paper is, very clearly, iterative and bottom-up. Using identified cycles in the activity graphs (eg. Figure 9, Linster and Musen, 1991) an initial inference pattern (see Figure 10, Linster and Musen, 1991) was constructed. This construction activity appeared to be guided, in a general fashion, by the KADS approach in the sense that 'roles' and 'intermediate results' were viewed as 'meta-classes' and 'activities' as 'knowledge sources'. Also the pattern in Figure 9 of the Linster and Musen paper conformed to a KADS format. The knowledge engineering task was to begin with the initial inference pattern and to continue to 'compare' each new classified pattern from the activity graph to the inference structure. When a new pattern would not 'fit' (be accounted for by) the existing inference structure, the inference structure was enlarged to accommodate the new pattern. At each step in the cycle new domain-independent terms were required and they were integrated conceptually into the interpretation structure. Once the interpretation model was fully developed, the cognitive processes of class or category assignment were required to select and slot a particular domain term into the appropriate area in the inference structure. The completed conceptual model contained all relevant domain knowledge along with the inference structure.

Based on the KADS requirements, a task and strategy level model was also required. In order to identify the task layer attributes, the knowledge engineer must describe how the problem solver organizes the sequence of knowledge sources (see Figures 18 and 19, Linster and Musen, 1991). This task model usually appears as a goal decomposition tree with knowledge sources from the inference layer placed to satisfy identified goal and sub-goal nodes. After a discussion with the expert, the knowledge engineer determined that a task decomposition was not necessary because the original ONCOCIN developers had already encripted a task decomposition- like structure into ONCOCI. At this point in the development of K-ONCOCIN, the original information sources were considered with this task structure in mind. The task decomposition in the Linster and Musen paper are descriptions of a likelt task decomposition stage had one been necessary. The knowledge engineer identified a useful tool, however, which would help identify and label task structures in terms of goal decomposition had one been necessary. No strategy was required because the task structure would not require dynamic modification (Linster and Musen, 1991).

Many of the cognitive processes evident in this stage have been outlined in the previous stage but apply here as well. Also, many of the same biasing processes may be evident in this stage as well. The knowledge engineer noted that he made no test to determine if he was correct and/or consistent in translating his labelled inference structures into KADS inference structures (see Table 3, Linster and Musen, 1991).

6.4 Outcomes

The most evident outcome of the cognitive activity in this stage is the new interpretation model for the KADS library and a final KADS conceptual model. More specifically, the result is a complete set of roles and actions which appear sufficient enough to explain the problem-solving activity in the domain.

The processes required to develop the conceptual model were briefly outlined as 'category assignment' and the development of the design model was not discussed in detail. Once the interpretation model was developed, much of the cognitive processing was accounted for and the selection of the formal representation was based on practical circumstances and experience rather than based on a demanding cognitive process. The knowledge engineer noted that " the selection of the techniques necessary will be strongly influenced by the anticipated environment (expert system shell)".

The knowledge engineer generated another principle from this stage of activity. When mapping from one epistemological system to another, *the meaning of a concept must be retained so that the concepts, within either system, are used with no loss of integrity.* This statement refers not only to the abstraction process from domain terms to the knowledge engineer's intermediate epistemological system, but again from this intermediate system to the KADS epistemological system.

7. Conclusions

7.1 Summary

By completing a focused interview with a knowledge engineer about a specific domain problem, it was possible to identify how the knowledge engineer viewed and described his tasks, the various solutions he attempted, the successes and failures of the solutions, and the classes of likely cognitive processes predominant in each stage of knowledge engineering. The interview provided insight into the level of functionality of the KADS modelling methodology and insight into the process of 'bottom-up' modelling. The interview was intended as a cursory or initial view of the cognitive activity of a knowledge engineer and the results are not intended as representative of all knowledge engineering activity.

7.2 Knowledge Engineering Tasks

The first three stage of knowledge engineering activity outlined in this paper reflect a variety of different tasks. The initial task of orientation to the domain required the knowledge engineer to 'make sense' of the knowledge structures he was given. This was no small task and considerable time and cognitive effort was required to obtain sufficient comprehension. The second stage of describing initial domain structure necessitated the identification or discovery of patterns between and among domain variables. The third stage required the development of an identifiable inference structure within the domain which essentially reflected the problem-solving process of the domain. Although the nature of the tasks demands at each stage overlapped to some extent, it was apparent that there were great differences in task demands at each stage. The knowledge engineer was faced with the challenging nature of these tasks as he moved through the stages of KADS model development.

From this single case study it is difficult to generalize the stage demarcations and the task descriptions to other knowledge engineering activity but the case study results do suggest that more study and description of the these various tasks is required. Very little information is available which describes knowledge engineering tasks . Although this paper reviewed a 'modelling approach' to knowledge engineering (and only a part of this activity), it is apparent that we lack a method for describing the various tasks faced by a knowledge engineer and consequently, we lack a body of knowledge useful for building knowledge-based systems.

7.3 Cognitive Processes in Knowledge Engineering

The successful completion of each stage of knowledge engineering required no small effort by the knowledge engineer. In most cases, different strategies and approaches were attempted in order to reach a solution to a problem or task. The different types of tasks, coupled with the available knowledge structures at each stage provided the 'stimulus set' which evoked the use of a variety of cognitive processes by the knowledge engineer. Classes of cognitive activity were identified and described along with research describing

the various cognitive processes in each class. The issue here is the amount of evidence for specific, identifiable cognitive processes used by the knowledge engineer while meeting the task demands. If the cognitive processes are evident and describable, then perhaps they are also supportable.

Again, it is difficult to generalize the results to other knowledge engineers by stating that the cognitive processes assumed in this case study would be evident for all other knowledge engineers. However, the knowledge engineer, like the domain novice, is faced with heavy cognitive demands. The processes needed to reduce the complexity of the domain into a useful form for knowledge engineers are clearly evident. Cognitive processes for pattern recognition, abstraction, pattern construction are likely involved in most knowledge engineering activity. What is needed is a much better elaboration, enumeration, and description of the cognitive processes of knowledge engineers as they meet the task demands of their craft. With this understanding more tools, techniques, and procedures can be developed to support the process.

This case study introduced the possibility of cognitive processes usually called 'biases' as part of the cognitive activity of the knowledge engineer. "Bias' is perhaps an unfortunate term because it connotes a 'problem' or an 'untrue' result. In any event, cognitive biases are cognitive processes. Reductive and judgmental biases were discussed at each stage. These cognitive processes were selected solely on the basis of the type of cognitive demands placed on the knowledge engineer by the tasks at each stage. If pattern recognition was a process then cue availability, frequency, and clarity were likely biasing or 'concomitant' cognitive processes. The truth or likelihood of this last statement requires testing and elaboration. Primary and concomitant cognitive processes of the knowledge engineer require further identification. In this case, the attention was placed on the cognitive processes needed to address the tasks in a 'bottom-up' approach to modelling within the KADS modelling methodology. Other knowledge engineering approaches also need a similar study.

7.4 Research Support for Addressing Cognitive Processes in Knowledge Engineering

This paper extend previous work (Woodward et al, 1991) in outlining how a number of models developed in cognitive science may be useful for knowledge engineering. The previous paper provided an overview of four areas of cognitive science research (text comprehension, judgment, social cognition, memory) which may have a direct impact on the development of tools, techniques and procedures for knowledge engineering. Many of the models in the cognitive science research require 'engineering' for use by knowledge engineers but the research support for many of the models is extensive. From this case study two separate tools were identified. The first was a tool which simply provided a library of domain-independent terms and how they had been applied previously by the knowledge engineer. This tool could be used to overcome some of the potential 'biases' during the appropriate stage. A second tool was briefly mentioned which addresses how a domain might be decomposed into task and sub-tasks in a manner useful to the knowledge engineer; a tool not primarily used for the manipulation of knowledge structures.

7.5 KADS and 'Bottom-up" Modelling

The approach taken by the knowledge engineer in this case study has been referred to as 'bottom-up' modelling for obvious reasons. Even though the KADS modelling approach was incorporated, its structures and its functionality did not significantly 'kick-in' until the end of the bottom-up modelling process. It is noted by Linster and Musen (1991) that the KADS method has its strengths and weaknesses. For bottom-up modelling, the KADS

approach provided increasingly more structure and guidance after an initial inference layer was developed. The usefulness of KADS appears to be in the concept of layers or models and in the structural definition of the interpretation model. These two concepts are useful guides and may provide a solid basis for eventual standards of modelling in knowledge engineering. However, the approach offers little detailed help in the first three stages identified in this case study. It may be concluded that a good deal of bottom-up modelling may need to occur even though KADS is a chosen knowledge engineering strategy.

The knowledge engineer stressed the cyclic aspects of this form of modelling. He reported that the attention paid to the domain view was always followed by attention paid to the inference view, followed again by attention to the domain view. This cycling activity ceased when all observable phenomena were covered in a satisfactory model. The cycling activity was necessary to integrate the two separate sets of epistemological structures.

The knowledge engineer does not simply select a model or identify a task and select a tool, technique, or procedure to deal with the task. There is a great deal of trial and error and a great deal of creativity involved. How did the knowledge engineer know when to change strategies; what other approaches were available to him of which he was unaware; how did he initially develop the informal inference structures; what gave him the idea to use the terms he did in his intermediate model before proceeding to KADS terms? How much knowledge and understanding of the domain did the knowledge engineer need before he could develop the model K-ONCOCIN?

The Linster and Musen (1991) paper represents an extremely important contribution to knowledge engineering practice. Its conception formed the goal of converting an established system into a popular knowledge engineering modelling system, guided an interesting and inventive knowledge engineering process, and provided an extremely useful exemplar of convergence of approaches. This case study has offered a 'cognitive' perspective of the Linster and Musen paper and should contribute to a better understanding of how K-ONCOCIN was developed.

References

Baker, L. and Brown, A.L. (1984a) Cognitive monitoring in reading. In J. Flood (Ed.) **Understanding Reading Comprehension: Cognition, Language and the Structure of Prose** (pp. 21-44). DE: International Reading Association: Newark.

Britton, B.K.,Van Dusen, L., Glynn, S.M., and Hemphill, D. (1990) The impact of inferences on instructional text. In Graesser, A.C. and Bower, G.H. (Eds.), **Inferences and Text Comprehension** (pp. 53-70). Academic Press: New York.

Brown, A.L., Armbruster, B.B., and Baker, L. (1986) The role of metacognition in reading and studying. In J. Orasanu (Ed.), **Reading Comprehension: From Research to Practice** (pp. 49-75). Erlbaum: Hillsdale, N.J.

Cleaves, D.A. (1986). Cognitive biases and corrective techniques: proposals for improving elicitation procedures for knowledge-based systems. In proceedings of the **First AAAI Knowledge Acquisition Workshop for Knowledge-Based Systems**: Banff, Canada.

Coulson, R.J., Feltovich, P.J., and Spiro, R.J. (1986) Foundations of a Misunderstanding of the Ultrastructural Basis of Myocardial Failure: A Reciprocating Network of Oversimplifications. **Report #1 Conceptual Knowledge Research Project**, Southern Illinois University School of Medicine, Springfield, IL

Estes, W.K. (1976) The cognitive side of probability learning. **Psychological Review**, 83, 37-64.

Estes, W.K. (1986) Array models for category learning. **Cognitive Psychology**, 18, 500-549.

Estes, W.K. (1989) Early and late memory processing in models for category learning. In C. Izawa (Ed.) **Current Issues in Cognitive Processes: The Tulane Flowertree Symposium on Cognition** (pp. 11-24). LEA: Hillsdale, N.J.

Feltovich, P.J., Spiro, R.J., and Coulson, R.L. (1989). The nature of conceptual understanding in biomedicine: the deep structure of complex ideas and the development of misconceptions. In D.A. Evans and V.L. Patel (Eds.), **Cognitive Science on Medicine: Biomedical Modeling** (pp. 113-172). MIT Press: London.

Feltovich, P.J., Johnson, P.E., Moller, J.H., and Swanson, D.B. (1984) LCS: the role and development of medical knowledge in diagnostic expertise. In W.J. Clancey and E.H. Shortcliffe (Eds.) **Readings in medical Artificial Intelligence: The First Decade**. Addison-Wesley: Reading, MA.

Garner, R. (1988). Strategies for reading and studying expository text. **Educational Psychologist**.

Gelman, S.A. and Markman, E. (1986). Categories and induction in young children. **Cognition**, 23, 183-209.

Hogarth, R. (1987) **Judgment and Choice (2nd Ed)**. Wiley: New York.

Hogarth, R., and Makridakis, S. (1981) Forecasting and planning: an evaluation. **Management Science**, 27(2).

Kahneman, D.L. and Tversky, A. (1973). On the psychology of prediction. **Psychological Review**, 80, 273-251.

Kahneman, D.L., Slovic, P. and Tversky, A. (Eds) (1982) **Judgment Under Uncertainty: Heuristics and Biases**. Cambridge University Press: New York.

Keeney, R.L. (1982) Decision analysis: an overview. **Operations Research**, 30, 803-838.

Lesgold, A.M. and Perfetti, C.A. (1978) Interactive processes in reading comprehension. **Discourse Processes**, 1, 323-336.

Linster, M. and Musen, M (1991). K-ONCOCIN: the use of KADS to model the task ONCOCIN. In proceedings of the **Sixth AAAI Knowledge Acquisition Workshop for Knowledge-Based Systems**: Banff, Canada.

Margolis, H. (1987). **Patterns, Thinking, and Cognition**. University of Chicago Press: Chicago.

Markman, E.M. (1977) Realizing that you don't understand: elementary school children's awareness of inconsistencies. **Child Development**, 50, 643-655.

Meyer, M.A. and Booker, J.M. (1991). **Eliciting Expert Judgment**. Academic Press: New York.

Musen, M.A., FAgan, L.M., Combs, D.M., and Shortcliffe, E.H. (1987) Use of a domain-model to drive an interactive knowledge-editing tool. **International Journal of Man-Machine Studies**, 26, 105-121.

Smith, E.E. and Medin, D.L. (1981) **Categories and Concepts**. Harvard University Press: Cambridge, MA.

Smith, E.E. (1990). Categorization. In D.N. Osherson and E.E. Smith (Eds.), **Thinking** (pp. 33-54). MIT Press: Cambridge, MA.

Spiro, R.J., Feltovich, P.J., and Coulson, R.L. (1988) Seductive reductions: the hazards of oversimplification of complex concepts. **Report #4 Conceptual Knowledge Research Project**, Southern Illonois University School of medicine, Springfield, IL.

Spiro, R.J. (1980) Constructive processes in prose comprehension and recall. In R.J. Spiro, B.C. Bruce, and W.F. Brewer (Eds.), **Theoretical Issues in Reading Comprehension**. Lawrence Erlbaum Associates: Hillsdale, N.J.

Spiro, R.J., Feltovich, P.J., Coulson, R.L., and Anderson, D.K. (1990) multiple analogies for complex concepts: antidotes for analogy-induced misconceptions in advanced knowledge acquisition. In S. Vosniadou and A. Ortony (Eds.) **Similarity and Analogical Reasoning**. Cambridge University Press: Cambridge, UK.

van Dijk, T.A. and Kintsch, W. (1983). **Strategies of Discourse Comprehension**. Academic Press,: New York.

von Winterfeldt, D. and Edwards, W. (1986) **Decision Analysis and Behavioral Research**, Cambridge University Press: Cambridge. UK.

Wielinga, B.J., Schreiber, A. Th., Breuker, J.A. (1991) **KADS: A Modelling Approach to Knowledge Engineering**. ESPRIT Document KADS-II/T1.1/PP/UvA/008/1.0. University of Amsterdam.

Wetter, Th., and Woodward, J.B. (1990) Towards a theoretical framework for knowledge acquisition. In proceedings of the **Fifth AAAI Knowledge Acquisition Workshop for Knowledge-Based Systems**: Banff, Canada.

Woodward, J.B., Shaw, M.L.G., and Gaines, B.R. (1991) The Cognitive Basis of Knowledge Engineering. In Schmalhoffer, F., Wetter, Th., and Voss, A. (Eds) **Cognition and Knowledge Engineering**. Springer-Verlag, Germany.