Proceedings of the 5TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre
Banff, Alberta, Canada
November 4-9, 1990

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
5TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre, Banff, Alberta, Canada
November 4-9, 1990

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.

5TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence
Banff, Canada, November 6-11, 1988

CONTENTS

Program iv

List of Participants and Authors ix

Technical Papers (listed alphabetically by first author)

A knowledge acquisition framework based on conceptual modeling 1-1
Mihai Barbuceanu, Institute for Informatics

Acquiring and refining procedures according to context 2-1
Guy Boy, NASA-Ames Research Center

Measuring the Structure of Diagnostic Expertise using Pathfinder 3-1
John H. Bradley, East Carolina University; Lawrence L. Schkade, University of Texas at Arlington

Knowledge acquisition during interactive planning: Exploiting human-generated exceptions 4-1
Carol A. Broverman, W. Bruce Croft, University of Massachusetts

Handling conflict between domain descriptions with computer-supported negotiation 5-1
Steve M. Easterbrook, Imperial College of Science and Technology

Meta-tool support for customized domain-oriented knowledge acquisition 6-1
Henrik Eriksson, Linköping University

ICONKAT: An integrated constructivist knowledge acquisition tool 7-1
Kenneth M. Ford (University of West Florida), Howard Stahl (Cornell University), Jack Adams-Webber (Brock University), Joseph Novak (Cornell University), Jeremy C. Jones (University of West Florida)

An architecture for integrated knowledge acquisition systems 8-1
Brian Gaines, University of Calgary

Cognitive and logical foundations of knowledge acquisition 9-1
Brian R. Gaines, Mildred L. G. Shaw, University of Calgary

Envisoning Group Knowledge Acquisition: Focus Groups in the Context of a Mixed Knowledge Acquisition Methodology 10-1
Martha Grabowski, Anne Massey, William A. Wallace, Rensselaer Polytechnic Institute

The Use of Lotus Agenda in Knowledge Acquisition for an Expert System in a Poorly Controlled Domain 11-1
Anton R. Grashion, Rocky Mountain Banknote

Knowledge acquisition for spatial knowledge-based system 12-1
Kazuo Hiraki, Yuichir Anzai, Keio University

Models of Problem-Solving in Knowledge-Based Systems: Their Focus on Knowledge Acquisition 13-1
Werner Karbach, Marc Linster, Angi Voß, GMD

Usable and reusable programming constructs 14-1
Georg Klinker, Carlos Bhola, Geoffroy Dallemagne, David Marques, John McDermott, Digital Equipment Corp

Theory-based Transfer of Teleological Knowledge 15-1
Lawrence Lefkowitz, Dirk Mahling, W. Bruce Croft, University of Massachusetts

Hyper-KSE: A heterogeneous integration of acquisition, interpretation, and documentation tools 16-1
Marc Linster (GMD), Brian Gaines (University of Calgary)

Instructable systems 17-1
Bruce MacDonald, University of Calgary

Opportunistic acquisition of plan knowledge 18-1
Dirk E. Mahling, Oddmar A. Sandvik, University of Massachusetts

KATE: A Piece of Computer Aided Knowledge Engineering 19-1
Michel Manago (IntelliSoft), Yves Kodratoff (CNRS)

Interactive Acquisition of Design Decisions 20-1
William Mark, Jon Schlossberg (presenting), Lockheed AI Center

The ethnological method of participant observation as a technique for manual knowledge acquisition 21-1
Mary A. Meyer, Rick R. Picard, Los Alamos National Laboratory; Jeffrey M. Ross, Los Alamos Medical Center

A Knowledge Acquisition System for Analysing Prescriptive Texts 22-1
Bernard Moulin, Daniel Rousseau, Université Laval

Verification of frame and semantic network knowledge bases 23-1
Daniel E. O'Leary, University of Southern California

The method of direct knowledge acquisition from human experts 24-1
Gennadi S. Osipov, Program Systems Institute of the USSR Academy of Sciences

Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning 25-1
Michael J. Pazzani, Clifford A. Brunk, University of California

Design Knowledge Acquisition: Task Analysis and a Partial Implementation 26-1
Yoram Reich, Carnegie Mellon University

Automatic knowledge acquisition from texts: learning terminological knowledge via text understanding and inductive generalization 27-1
Ulrich Reimer, Universitat Konstanz

Transferring knowledge from an active expert to an end-user environment 28-1
Kristian Sandahl (Linköping University), Anders Brogren (Volvo Data), Lars Reshagen (Linköping University)

Integrated knowledge acquisition from expert memories, previously solved cases, and text 29-1
Franz Schmalhofer, Otto Kühn, Gabriele Schmidt, German Research Center for Artificial Intelligence

On problems with the knowledge level perspective 30-1
Guus Schreiber, Hans Akkermans, Bob Wielinga, University of Amsterdam

Acquiring knowledge for validation models 31-1
Evangelos Simoudis, James Miller, Digital Equipment Corporation

Ontological Issues in Knowledge Base Design: Some Problems and Suggestions 32-1
Doug Skuce (University of Ottawa), Ira Monarch, (Carnegie Mellon University)

Acquiring strategic knowledge in adversarial domains 33-1
Steven Walczak, University of Florida

In what sense does knowledge exist? Alternatives and consequences for knowledge engineering 34-1
Thomas Wetter, IBM Germany

Towards a theoretical framework for knowledge acquisition 35-1
Thomas Wetter (IBM Germany), Brian Woodward (University of Calgary)

A Knowledge-Acquisition Approach for Error Handling in Supervised Autonomous Robotic Systems 36-1
Y. Zheng, L. K. Daneshmend, McGill University

PROGRAM

5TH BANFF KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

In Cooperation with the American Association for Artificial Intelligence
Banff, Alberta, Canada, November 4-9, 1990

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

Program Committee

Jeffrey Bradshaw, Boeing Advanced Technology Center
B. Chandrasekaran, Ohio State University
William Clancey, Institute for Research on Learning
Michael J. Freiling, Tektronix, Inc.
Thomas Gruber, Stanford University
Catherine Kitto, Boeing Advanced Technology Center
Sandra Marcus, Boeing Commercial Airplane Group
John McDermott, Digital Equipment Corporation
Ryszard Michalski, George Mason University
Mark Musen, Stanford University
Bruce Porter, University of Texas at Austin
Mildred Shaw, University of Calgary

DAILY SCHEDULE

Workshop sessions will be held in the residence 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 a separate building. There will be group dinners Monday, Tuesday, and Thursday evenings in a separate Centre building.

Talks will last **40 minutes** - **20 minutes** for presentation, **5 minutes** for a response from a discussion leader, and **15 minutes** for discussion. Please keep strictly to the presentation time to allow ample time for discussion; do not force us to resort to the use of bears and sharks. **Panel** discussions will also last **40 minutes** - **10 minutes** each for 3 speakers and **10 minutes** of general discussion.

SCHEDULE OF EVENTS

Sunday, November 4

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

Monday, November 5

Overview

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

Integration

9:40 - 10:20	An architecture for integrated knowledge acquisition systems Brian Gaines, University of Calgary (discussion: Marc Linster)
10:20 - 10:40	Break

Attacking Synthesis Domains

10:40 - 11:20 Knowledge acquisition during interactive planning: Exploiting human-generated exceptions
Carol A. Broverman, W. Bruce Croft, University of Massachusetts (discussion: Dirk Mahling)

11:20 - 12:00 Interactive Acquisition of Design Decisions
William Mark, Jon Schlossberg (presenting), Lockheed AI Center (discussion: Debbie Leishman)

12:00 - 1:00 Lunch

Knowledge Acquisition from Text

1:00 - 1:40 A Knowledge Acquisition System for Analysing Prescriptive Texts
Bernard Moulin, Daniel Rousseau, Université Laval (discussion: Franz Schmalhofer)

Panel: Burning Issues in Knowledge Acquisition

1:40 - 2:20 John Boose, Boeing Advanced Technology Center
Enrico Motta, Open University
Bob Wielinga, University of Amsterdam

2:20 - 2:40 Break

Participant Observation

2:40 - 3:20 The ethnological method of participant observation as a technique for manual knowledge acquisition
Mary A. Meyer, Rick R. Picard, Los Alamos National Laboratory; Jeffrey M. Ross, Los Alamos Medical Center (discussion: Maurice Sharp)

Theory and Practice

3:20 - 4:00 In what sense does knowledge exist? Alternatives and consequences for knowledge engineering
Thomas Wetter, IBM Germany (discussion: Hugh Leighton)

7:00 Dinner buffet, full cash bar

Tuesday, November 6

Practical Aspects of Machine Learning

9:00 - 9:40 KATE: A Piece of Computer Aided Knowledge Engineering
Michel Manago (IntelliSoft), Yves Kodratoff (CNRS) (discussion: Kazuo Hiraki)

9:40 - 10:20 Detecting and correcting errors in rule-based expert systems: an integration of empirical and explanation-based learning
Michael J. Pazzani, Clifford A. Brunk, University of California (discussion: Ian Witten)

10:20 - 10:40 Break

Tools and Methods in Use

10:40 - 11:20 Meta-tool support for customized domain-oriented knowledge acquisition
Henrik Eriksson, Linköping University (discussion: Jukka Kontio)

Combining Models

11:20 - 12:00 Envisoning Group Knowledge Acquisition: Focus Groups in the Context of a Mixed Knowledge Acquisition Methodology
Martha Grabowski, Anne Massey, William A. Wallace, Rensselaer Polytechnic Institute (discussion: Steve M. Easterbrook)

12:00 - 1:00 Lunch

Tools and Methods in Use

1:00 - 1:40 Transferring knowledge from an active expert to an end-user environment
Kristian Sandahl (Linköping University), Anders Brogren (Volvo Data), Lars Reshagen (Linköping University) (discussion: Gennadi S. Osipov)

Panel: KA and Verification, Validation, and Maintenance: What is Practical?

1:40 - 2:20 Jasbir Dhaliwal, University of British Columbia
Daniel E. O'Leary, University of Southern California
Evangelos Simoudis, Digital Equipment Corporation

2:20 - 2:40 Break

Theory and Practice

2:40 - 3:20 Cognitive and logical foundations of knowledge acquisition
Brian R. Gaines, Mildred L. G. Shaw, University of Calgary (discussion: BobWielinga)

3:20 - 4:00 Ontological Issues in Knowledge Base Design: Some Problems and Suggestions
Doug Skuce (University of Ottawa), Ira Monarch, (Carnegie Mellon University) (discussion: Bruce MacDonald)

7:00 Dinner, full cash bar

Wednesday, November 7

Theory and Practice

9:00 - 9:40 On problems with the knowledge level perspective
Guus Schreiber, Hans Akkermans, Bob Wielinga, University of Amsterdam (discussion: Yoram Reich)

9:40 - 10:20 Models of Problem-Solving in Knowledge-Based Systems: Their Focus on Knowledge Acquisition
Werner Karbach, Marc Linster, Angi Voß, GMD (discussion: Brian Woodward)

10:20 - 10:40 Break

Panel: Human Aspects of Knowledge Acquisition

10:40 - 11:20 Jack Adams-Webber, Brock University
Robert Hoffman, Adelphi University
Brian Woodward, University of Calgary

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

Thursday, November 8

Attacking Synthesis Domains

9:00 - 9:40 Acquiring and refining procedures according to context
Guy Boy, NASA-Ames Research Center (discussion: Y. Zheng)

9:40 - 10:20 Usable and reusable programming constructs
Georg Klinker, Carlos Bhola, Geoffroy Dallemagne, David Marques, John McDermott, Digital Equipment Corp (discussion: Jasbir Dhaliwal)

10:20 - 10:40 Break

Psychometric Tools

10:40 - 11:20 ICONKAT: An integrated constructivist knowledge acquisition tool
Kenneth M. Ford (University of West Florida), Howard Stahl (Cornell University), Jack Adams-Webber (Brock University), Joseph Novak (Cornell University), Jeremy C. Jones (University of West Florida) (discussion: John H. Bradley)

11:20 - 12:00 Measuring the Structure of Diagnostic Expertise using Pathfinder
John H. Bradley, East Carolina University; Lawrence L. Schkade, University of Texas at Arlington (discussion: Kenneth M. Ford)

12:00 - 1:00 Lunch

Working Groups

Suggested Working Groups

Afternoon

Advanced Architectures (reporting: Marc Linster, Mildred Shaw)
Combining Multiple Methods, Representations, and Expertise (Reporting: Jeff Bradshaw, Martha Grabowski)
Integration of Machine Learning and Knowledge Acquisition (reporting: Michel Manago, Ian Witten)
KA and Verification, Validation, and Maintenance (reporting: Daniel O'Leary, Bob Wielinga)
Practical Tools from Psychology (reporting: Jack Adams-Webber, Mary Meyer)
Knowledge Acquisition: The Next 10 Years (reporting: Tom Addis, Brian Woodward)

2:20 - 2:40 Break

7:00 Banquet Dinner with Annual Bear Tossing and Shark Snatching Mystery Guest Speaker

Friday, November 9

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 and Bear Souvenirs; Business Meeting

POSTERS

Attacking Synthesis Domains

Opportunistic acquisition of plan knowledge
Dirk E. Mahling, Oddmar A. Sandvik, University of Massachusetts

Design Knowledge Acquisition: Task Analysis and a Partial Implementation
Yoram Reich, Carnegie Mellon University

Acquiring strategic knowledge in adversarial domains
Steven Walczak, University of Florida

Combining Models

Handling conflict between domain descriptions with computer-supported negotiation
Steve M. Easterbrook, Imperial College of Science and Technology

Integration

Integrated knowledge acquisition from expert memories, previously solved cases, and text
Franz Schmalhofer, Otto Kühn, Gabriele Schmidt, German Research Center for Artificial Intelligence

KA and Verification, Validation, and Maintenance

Acquiring knowledge for validation models
Evangelos Simoudis, James Miller, Digital Equipment Corporation

Verification of frame and semantic network knowledge bases
Daniel E. O'Leary, University of Southern California

Knowledge Acquisition from Text

Automatic knowledge acquisition from texts: learning terminological knowledge via text understanding and inductive generalization
Ulrich Reimer, Universitat Konstanz

Practical Aspects of Machine Learning

Knowledge acquisition for spatial knowledge-based system
Kazuo Hiraki, Yuichir Anzai, Keio University

Instructable systems
Bruce MacDonald, University of Calgary

A Knowledge-Acquisition Approach for Error Handling in Supervised Autonomous Robotic Systems
Y. Zheng, L. K. Daneshmend, McGill University

Theory and Practice

Theory-based Transfer of Teleological Knowledge
Lawrence Lefkowitz, Dirk Mahling, W. Bruce Croft, University of Massachusetts

A knowledge acquisition framework based on conceptual modeling
Mihai Barbuceanu, Institute for Informatics

Towards a theoretical framework for knowledge acquisition
Thomas Wetter (IBM Germany, Scientific Center), Brian Woodward (University of Calgary)

Tools and Methods in Use

The Use of Lotus Agenda in Knowledge Acquisition for an Expert System in a Poorly Controlled Domain
Anton R. Grashion, Rocky Mountain Banknote

Hyper-KSE: A heterogeneous integration of acquisition, interpretation, and documentation tools
Marc Linster (GMD), Brian Gaines (University of Calgary)

The method of direct knowledge acquisition from human experts
Gennadi S. Osipov, Program Systems Institute of the USSR Academy of Sciences

LIST OF PARTICIPANTS AND AUTHORS

5th BANFF
KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff, Canada, November 4-9, 1990

H. Akkermans
Department of Social Science Info.
University of Amsterdam
Herengracht 196
1016 BS Amsterdam, The Netherlands

Yuichir Anzai
Dept of Computer Science
Keio University
3-14-1, Hiyoshi
Yokohama, 233 Japan

Mihai Barbuceanu
Institute for Informatics
8-10 Miciurin
71316 Bucharest 1 Romania

Carlos Bhola
Digital Equipment Corp
111 Locke Dr
Box 1006
Marlbouough, MA 01752 USA

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

John H. Bradley
Department of Decision Science
East Carolina University
Greenville, NC 27858 USA

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

Anders Brogren
Volvo Data
Dept 2630 DLI
S-405 08 Göteborg, Sweden

Carol Broverman
Dept of Computer and Information Science
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003 USA

Clifford A. Brunk
Dept of Information and Computer Science
University of California
Irvine, CA 92717 USA

W. Bruce Croft
Dept of Computer and Information Science
Lederle Graduate Research Center
University of Massachusetts
Amherst, MA 01003 USA

Geoffroy Dallemagne
Digital Equipment Corp
111 Locke Dr
Box 1006
Marlbouough, MA 01752 USA

Professor Daneshmend
McGill Research Centre for Intelligent Machines &
Dept. of Electrical Engineering
McGill University
McConnell Engineering Building
3480 University St
Montreal, Quebec, CANADA, H3A 2A7

Jasbir Dhaliwal
Faculty of Commerce and Business Administration
University of British Columbia
2053 Main Mall
Vancouver, B.C., Canada V6T 1Y8

Steve M. Easterbrook
Dept of Computing
Imperial College of Science and Technology
University of London
180 Queen's Gate, South Kensington
London SW7 2BZ UK

Henrik Eriksson
Dept of Computer and Information Science
Linköping University
S-581 83 Linköping, Sweden

Kenneth M. Ford
Division of Computer Science
University of West Florida
11000 University Parkway
Pensacola, FL 32514 USA

Brian Gaines
Knowledge Science Institute
Dept of Computer Science
University of Calgary
2500 University Dr NW
Calgary, Alberta, Canada T2N 1N4

Martha Grabowski
Dept of Decision Sciences and Engineering Systems
Center for Industrial Innovation 8211
Rensselaer Polytechnic Institute
Troy, NY 12180 USA

Anton R. Grashion
Rocky Mountain Banknote
10455 West 6th Ave
Lakewood, CO 80215 USA

Kazuo Hiraki
Dept of Computer Science
Keio University
3-14-1, Hiyoshi
Yokohama, 233 Japan

Jeremy C. Jones
Division of Computer Science
University of West Florida
11000 University Parkway
Pensacola, FL 32514 USA

Werner Karbach
GMD
Postfach 1240
Scholß Birlinghoven
D-5205 Sankt Augustin 1
West Germany

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

Yves Kodratoff
George Mason University and
CNRS
LRI, bât 490
Université de Paris sud
91405, ORSAY, Cedex, France

Jukka Kontio
NOKIA Research Center
PO Box 156
0201 Espoo
Finland

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

Lawrence Lefkowitz
Collaborative Systems Laboratory
Dept of Computer and Information
Science
305 Lederle Graduate Research
Center
University of Massachusetts
Amherst, MA 01003 USA

Hugh Leighton
Knowledge Sciences Institute
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada T2N
1N4

Debbie Leishman
NRC IRC
Calgary, Alberta

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

Bruce MacDonald
Knowledge Sciences Institute
Computer Science Dept
University of Calgary
2500 University Dr NW
Calgary, Alberta, Canada T2N
1N4

Dirk Mahling
Collaborative Systems Laboratory
Dept of Computer and Information
Science
305 Lederle Graduate Research
Center
University of Massachusetts
Amherst, MA 01003 USA

Michel Manago
16 Passage Foubert
75013, Paris, France

William Mark
Lockheed AI Center
3251 Hanover St
96-20 B/259
Palo Alto, CA 94304 USA

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

Anne Massey
Dept of Decision Sciences and
Engineering Systems
Center for Industrial Innovation
8211
Rensselaer Polytechnic Institute
Troy, NY 12180 USA

John McDermott
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

Ira Monarch
Philosophy Dept
Laboratory for Computational
Linguistics
Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA 15213 USA

Enrico Motta
Human Cognition Research
Laboratory
The Open University
Milton Keynes, MK7 6AA,
U.K.

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

Daniel E. O'Leary
Graduate School of Business
University of Southern California
Los Angeles, CA 90089 USA

Gennadi S. Osipov
Laboratory of Knowledge
Representation Problems
Program Systems Institute of the
USSR Academy of Sciences
152140, Pereslavie-Zalessky,
USSR

Michael J. Pazzani
Dept of Information and Computer
Science
University of California
Irvine, CA 92717 USA

Rick R. Picard
MS F600
Statistics Group
Los Alamos National Laboratory
Los Alamos, NM 87545 USA

Yoram Reich
Dept of Civil Engineering
Carnegie Mellon University
Pittsburgh, PA 15213 USA

Ulrich Reimer
Universitat Konstanz
Informationswissenschaft
Postfach 5560
D-7750 Konstanz, FRG

Lars Reshagen
Dept of Med Informatics
Linköping University
S-581 83 Linköping, Sweden

Jeffrey M. Ross
Los Alamos Medical Center

Los Alamos, New Mexico 87544
USA

Kristian Sandahl
Dept of Comp. and Infor Science
Linköping University
S-581 83 Linköping, Sweden

Oddmar A. Sandvik
Collaborative Systems Laboratory
Dept of Computer and Information Science
University of Massachusetts
305 Lederle Graduate Research Center
Amherst, MA 01003 USA

Lawrence L. Schkade
Information Systems Department
University of Texas at Arlington
Arlington, TX 76019

Jon Schlossberg
Lockheed AI Center
3251 Hanover St
96-20 B/259
Palo Alto, CA 94304 USA

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

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

Guss Schreiber
Dept of Social Science Informatics
University of Amsterdam
Herengracht 196
NL-1016 BS Amsterdam The Netherlands

Maurice Sharp
Knowledge Sciences Institute
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada T2N 1N4

Mildred L. G. Shaw
Knowledge Science Institute
Dept of Computer Science
University of Calgary
2500 University Dr NW
Calgary, Alberta, Canada T2N 1N4

Evangelos Simoudis
Digital Equipment Corporation
290 Donald Lynch Blvd
Marlboro, MA 01752 USA

Doug Skuce
Dept of Computer Science
University of Ottawa
Ottawa, Ontario, Canada

Howard Stahl, Joseph Novak
Cornell University

Angi Voß
GMD
Postfach 1240
Scholß Birlinghoven
D-5205 Sankt Augustin 1
West Germany

Steven Walczak
Computer and Information Sciences
Room 301, CSE Building
University of Florida
Gainesville, FL 32611 USA

William A. Wallace
Dept of Decision Sciences and Engineering Systems
Center for Industrial Innovation 8211
Rensselaer Polytechnic Institute
Troy, NY 12180 USA

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

Bob Wielinga
Department of Social Science Info.
University of Amsterdam
Herengracht 196
1016 BS Amsterdam, The Netherlands

Ian Witten
Knowledge Sciences Institute
Department of Computer Science
The University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada T2N 1N4

Brian Woodward
Knowledge Science Institute
Dept of Computer Science
University of Calgary
2500 University Dr NW
Calgary, Alberta, Canada T2N 1N4

Y. Zheng
McGill Research Centre for Intelligent Machines & Dept. of Electrical Engineering
McGill University
McConnell Engineering Building
3480 University St
Montreal, Quebec, CANADA, H3A 2A7

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

THE MODELS ENVIRONMENT FOR KNOWLEDGE ACQUISITION: AUTOMATING THE TASK-SPECIFIC ARCHITECTURE APPROACH

Mihai Barbuceanu

Expert Systems Research Laboratory
Institute for Informatics
8-10 Miciurin, 71316 Bucharest 1
Romania

ABSTRACT

We present the design of a knowledge acquisition environment supporting the task-specific architecture approach. The environment is based on a formal Conceptual Modeling Language able to describe the great variety of knowledge representation and processing mechanisms employed by task-specific architectures. The notion of task-specific architecture is formalized as an Abstract Knowledge Proccesing Machine whose inputs, outputs and internal mechanism are described in the Conceptual Modeling Language. A two stage approach is proposed for producing knowledge based systems: the modeling stage builds the conceptual model and the generation stage compiles it into an efficiently executable form. Generated systems consist of a knowledge acquisition tool, an inference engine and an initial knowledge base. The approach is illustrated with examples concerning modeling, integration and evolution of several task-specific architectures.

INTRODUCTION

Task-specific architectures are an increasingly important approach to knowledge acquisition and expert system construction. They formalize reccuring knowledge processing patterns which are both epistemologically adequate - simulating "natural" patterns of information processing - and computationally tractable - such patterns being necessarily efficient to compute.

The two most important lines of research in the task-specific vein are those of Chandrasekaran (1986, 1987) and McDermott (1988). The former regards task-specific architectures as problem/method/knowledge/inference packages (Chandrasekaran 1989) called generic tasks. Generic tasks model higher level aspects of cognition described e.g. as "hierarchical classification", "abstract plan instantiation and refinement", "hierarchical abstraction of data" and supported by high level tools like CSRL (Shum and Davis 1986) or DSPL (Herman, Josephson and Hartung 1986). The latter takes a more idiosyncratic view, developing problem solving methods specified as "propose-and-revise" (Marcus, McDermott and Tianran 1985), "acquire-and-present" (Klinker 1988) or "cover-and-differentiate" (Eshelman and McDermott 1986). These methods are used by task-specific knowledge acquisition tools to elicit, debug and maintain knowledge bases.

Up to now, the task-specific view has been manually applied to build isolated systems for various applications. The next step should be that of providing a methodology and an environment for automating the application of the task-specific approach to knowledge acquisition and expert system construction. This paper presents a preliminary design of such an environment, named MODELS.

The MODELS environment is based on the following ideas.

1. Since task-specific architectures develop specific knowledge representation and processing mechanisms, there must exist a unified formalism for describing these mechanisms. MODELS provides a formal Conceptual Modeling Language (CML) for this purpose.

2. Task-specific architectures are formalized as Abstract Knowledge Processing Machines (AKPM-s). The inputs, outputs and internal mechanism of AKPM-s are described in CML. AKPM-s can be assembled from other AKPM-s. They can be both simulated at the

knowledge level and compiled into efficient code in usual AI languages. AKPM-s are stored in the knowledge base of the environment.

3. Knowledge based system construction takes place in two steps. The first is a modeling step where the informal problem specifications coming from domain experts are converted into an AKPM. This AKPM is formal but potentially inefficient. The second step, generation, compiles the conceptual model represented by the AKPM into an executable AI representation language obtaining an efficient version of the AKPM. The resulting knowledge based system (a task-specific architecture) consists of a knowledge acquisition tool, an initial knowledge base and an inference engine.

The CML plays a central role in this framework. First, it provides a uniform desription language for the various task-specific architectures. Second, it makes it possible to build general support tools useable for all task-specific architectures. These include interpreters, validation tools, tools for maintenance, explanation, debugging, tools for generating the knowledge elicitation tools, etc. Without this uniform description these tools would had to be replicated for each architecture in part.

The MODELS approach is presented as follows. Section 2 gives an overview of the environment. Section 3 presents the CML. Section 4 discusses the AKPM concept. Section 5 presents the modeling step of the approach, and section 6 the generation step.

OVERVIEW OF THE APPROACH

Our goal is to produce a method and an environment which would accept as input a specification of a class of problems supplied by a domain expert and produce as output a task-specific knowledge based problem solving architecture able to solve problems from the intended class. The knowledge based problem solving architecture produced with the approach is composed of the blocks and interacts with the users shown in figure 1.

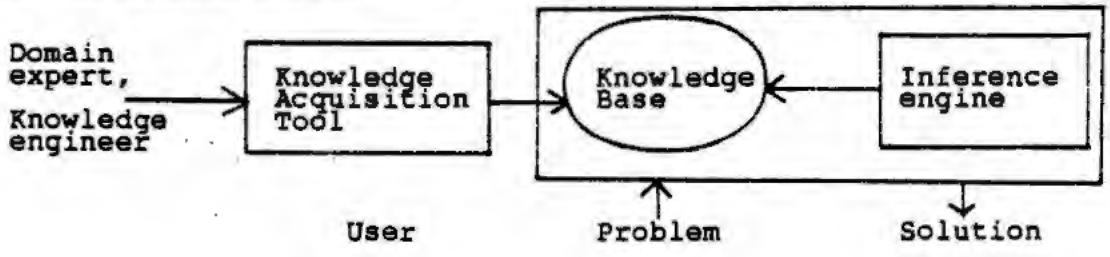


Figure 1: Knowledge based problem solving architecture

The method used by MODELS to produce this architecture is shown in figure 2. It divides the overall task in two steps, Modeling and Generation ((Twine 89) analyses this sort of division in detail). The first implements the transition from an informal specification supplied by human experts to a formal model of the architecture. The step is essentially heuristic, involving the domain expert and the knowledge engineer who must decide on the adequacy of the formal model. This model is a conceptual model encoded in terms of CML. It generally consists of an assembly of AKPM-s taken from the KB of the environment. The heuristic guidance provided in this phase relies on the existence in the MODELS knowledge base of a repertoire of AKPM-s, on problem-type taxonomies and on a general conceptual taxonomy serving as a shared background for communication.

The second step, Generation, transforms the clear but inefficient conceptual model into a less clear but more efficient system coded in a state-of-the-art AI language such as KEE (Fikes and Kehler 1985) or SRL (Wright and Fox 1984). Unlike Modeling, this step is essentially formal. It involves the knowledge engineer for specifying a number of details, but not for major decisions. Both steps have validation components, but these will not be dealt with in this paper.

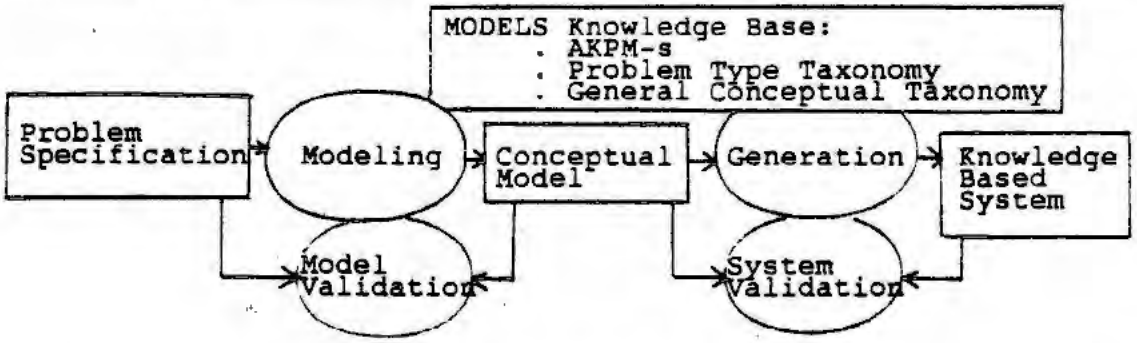


Figure 2: The MODELS approach

The CML is at the heart of the approach. Its purpose is that of coping with the diversity and ad-hoc-ness of task-specific architectures by providing a single description formalism in terms of which everything can be described. CML allows describing three kinds of entities. The first is the vocabulary of structured terms and concepts specific to the domain. The second is the relevant relationships among concepts and terms. The third is the problem solving activities performed upon terms and relationships. CML expresses the vocabulary of structured terms and concepts by means of a structured description language with formal semantics based on subsumption. This allows the definition and management of domain specific conceptual taxonomies. Relationships are described using a constraint language. Special mechanisms based on constraint propagation and abstraction levels decomposition are provided for specifying the activities performed during problem solving. Figure 3 illustrates the three basic ingredients of CML.

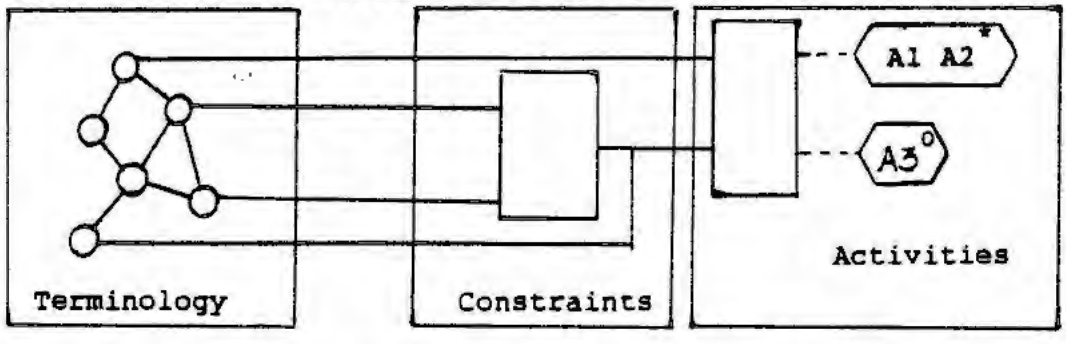


Figure 3: The CML components

The MODELS approach also formalizes the notion of task-specific problem solving architecture. The formal device capturing its meaning is the Abstract Knowledge Processing Machine (fig. 4). An AKPM is characterized by its inputs, its outputs and its internal mechanism. The inputs and outputs are specified as CML concepts and constraints. The internal mechanism is specified using internal concepts, constrains and CML activities.

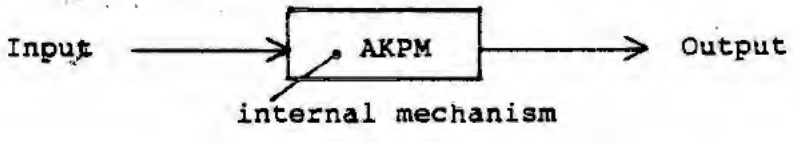


Figure 4: AKPM components

AKPM-s are also characterised by the abstraction level at which they are used. For example, given the KADS (Breuker and Wielinga 1987) abstraction level decomposition of knowledge based systems, we can have different AKPM-s at the domain, inference, task or strategy levels of a given system. Multi-level assemblies of AKPM-s form the Conceptual Model of the approach.

3. THE CONCEPTUAL MODELING LANGUAGE

We set the following requirements for the conceptual modeling language used to describe task-specific problem solving models:

1. Formal semantics for clarity and uniformity in describing multiple models;
2. Ability to describe both the static (i.e. concepts and relationships) and the dynamic (i.e. activities, processes) aspects of a domain model (Alexander et.al. 1986);
3. Ability to mechanically interpret (execute) the conceptual model and thus simulate the model directly at the knowledge level.

A conceptual model has three components: a taxonomy of structured concepts specifiying the terminology of the domain, a collection of constraints amongst them and a collection of activities involving and affecting the concepts and relationships.

3.1. Terminology

To describe the structure and organization of conceptual entities we have designed a structured description language with formal semantics, similar to terminological languages like KLONE (Brachman and Schmolze 1985), NIKL (Schmolze 1985), KRYPTON (Brachman, Gilbert and Levesque 1985), KANDOR (Patel-Schneider 1984), LOOM (MacGregor and Bates 1987) or BACK (Luck et.al. 1987).

Terminological knowledge is described by defining concepts and roles. Concepts correspond to classes of individuals while roles denote two-place relationships between individuals. The class of individuals denoted by a concept is called the extension of the concept. Essential for describing taxomomies of concepts is the subsumption relatinship: one concept subsumes another iff the extension of the first is a superset of the extension of the second.

Terminological languages - ours included - define the extension of concepts formally. This has the advantage of formalizing the semantics of the language which is an important asset in conceptual modeling due to the clarity it brings.

As the space does not allow us to enter details concerning the complete syntax and the formal semantics of such languages, we illustrate the major points by means of examples. Figure 5 shows an example taxonomy of concepts dealig with vehicles and cars. Figure 6 shows some concepts from the taxonomy illustrating some of the syntactic constructs available for specifying concepts and roles. For example, to describe the class of Ferrari cars we use the description:

```
[Ferrari  (and [a Car
                motor [a V12Engine]]
               [a Possesion ])
  Manufacturer "Ferrari"].
```

This creates the concept Ferrari as a conjunction of concepts Car and Possesion. As such, this concept possesses all roles coming from Car and Possesion to which it adds role Manufacturer. The Car concept used in the conjunction is actually a specialization of the Car concept, being restricted to have only V12Engine-s.

Besides conjunction and specialization the description language provides other concept and role forming operators such as cardinality restrictions (used to constrain the number of fillers of a role as in figure 6), subroles, filtering, etc. The language distinguishes primitive concepts which contain only necessary conditions, from terminal (defined) concepts which contain both necessary and sufficient conditions and which can not be further specialized.

The major service provided by terminological languages is the computation of subsumption. Given two concepts A and B, the language provides a predicate subsumes (A B) which tells if A subsumes B, that is iff all individuals of class B also belong to class A. Being able to evaluate this predicate for any two concepts is at the basis of building taxonomies of concepts. Subsumption depends exclusively on the structure of concepts (roles and their fillers) and not on their names.

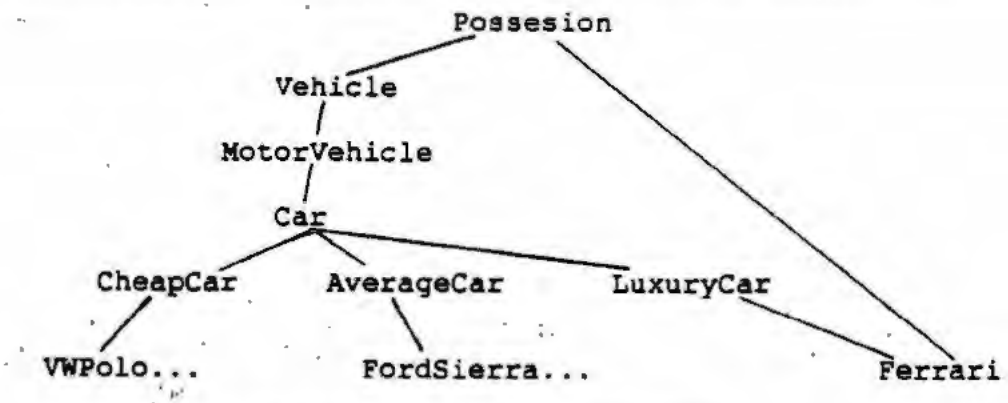


Figure 5: Concept taxonomy

```
[Possesion
   owner [a Person]
   resale-value [a Value unit $]]

[Vehicle [a Possesion]
   driver (all Human atleast 1)

   speed [a Value unit mph]
   #wheels [a NatNum]]
[MotorVehicle [a Vehicle]
   motor [a CombustionEngine]]

[Car [a MotorVehicle
        #wheels 4]
   miles-per-gallon [a Value
                     unit mile]
   mileage [a Value unit mile]
   fuel-quantity [a Value
                  unit gallon]
   distance [a Value unit mile]
   price [a Value unit $]]
```

Figure 6: Concepts and roles

The interested reader may find in the previously cited literature many details concerning these languages and the computation of subsumption (Brachman and Levesque 1984, Nebel 1988). We only mention that subsumption is often expensive to compute and for this reason we use a simpler description language and compensate for the lack of descriptive power by using other modeling mechanism such as the constraints and activity specifications to be presented next.

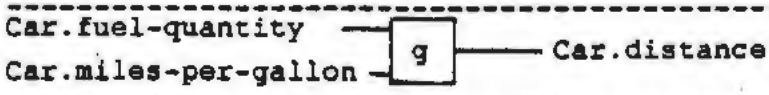
3.2 Constraints

Concept taxonomies are good for specifying the definitional aspects of the concepts involved. We also need to describe other relationships and dependencies among concepts in the vocabulary. For this purpose we use constraints.

A constraint is a n-place predicate among concepts in the taxonomy. Each argument of the predicate is typed, that is it must belong to the class described by a certain concept or role.

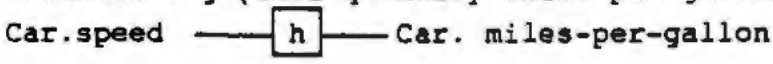
Figure 7 illustrates some constrains in the vehicles taxonomy. These constraints link the possible values of some roles belonging to the Car concept. For example, the distance a car can cover is constrained by its fuel-quantity and miles-per-gallon.

Constraints can be also defined by composing other constraints, as is the case with the constraint linking the distance with speed and fuel-quantity.



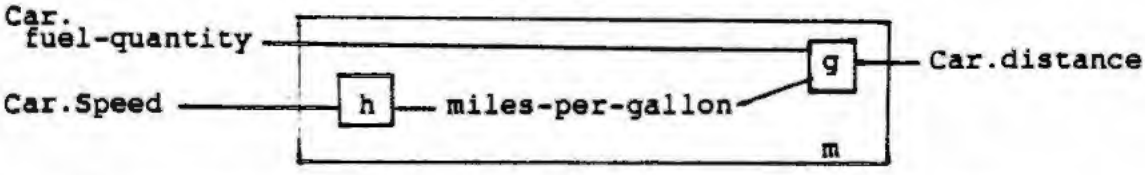
a) The distance a car can cover depends on the quantity of fuel in the tank and the miles.per-gallon:

distance = g (fuel-quantity miles-per-gallon)



b) The miles-per-gallon depends on the speed:

miles-per-gallon = h (speed)



c) The distance also depends on the speed and fuel-quantity (by composing a and b)

Figure 7: Constraints

For reasons to be detailed next, it is often useful to express a constraint as an exclusive or of several predicates. These components are called cases. For example, the constraint that two variables have distinct values can be expressed by means of two cases, bigger and smaller, their defining predicates being those in figure 8.

```
DistinctValues
 arguments
     x: Number
     y: Number
 cases
     bigger  (x > y)
     smaller (x < y)
```

Figure 8: Constraint with cases

Constraints can be expressed using the same structured description language, and organized into taxonomies. Cases will appear as roles filled with the defining predicates. Figure 9 shows the DistinctValues constraint in structured concept form, as well as a small taxonomy of constraints.

```
[DistinctValues
 arguments ( x y)
 x [a Number]
 y [a Number]
 bigger (> x y)
 smaller (< x y)]
```

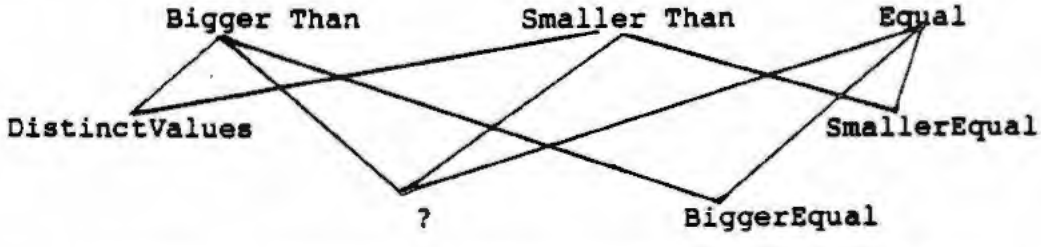


Figure 9: Structured representation of constraints and constraint taxonomy

3.3 Activities

The terminology and in part the constraints define what has been called (Alexander et.al. 1986) the static part of the universe of discourse. These are the elements which remain more or less stable in the course of problem solving. The conceptual model must also account for the processes or activities which change the universe of discourse during problem solving. We achieve this by means of two specific formalism, the Constraint Activity Network (CAN) and the Constraint Procedures (CP).

3.3.1 Constraint Activity Networks

CAN-s are usual constraint networks augmented with activities which are carried out during constraint propagation. Activities are attached to individual constraints by specifying the cases or the case transitions in which the activities must be executed.

Figure 10 shows a CAN which decides wheather a given car can reach town B starting from town A in n hours. It consists of a constraint network augmented with two activities, Accept and Reject. The first is executed when constraint "?" (which merely compares two quantities) is in one of the states bigger-than and equal, and the second when the constraint is in state less-than. In general, the specification of the situations when attached activities must be executed includes disjunctions of cases (as in the example) negation of cases and transitions from one case to another.

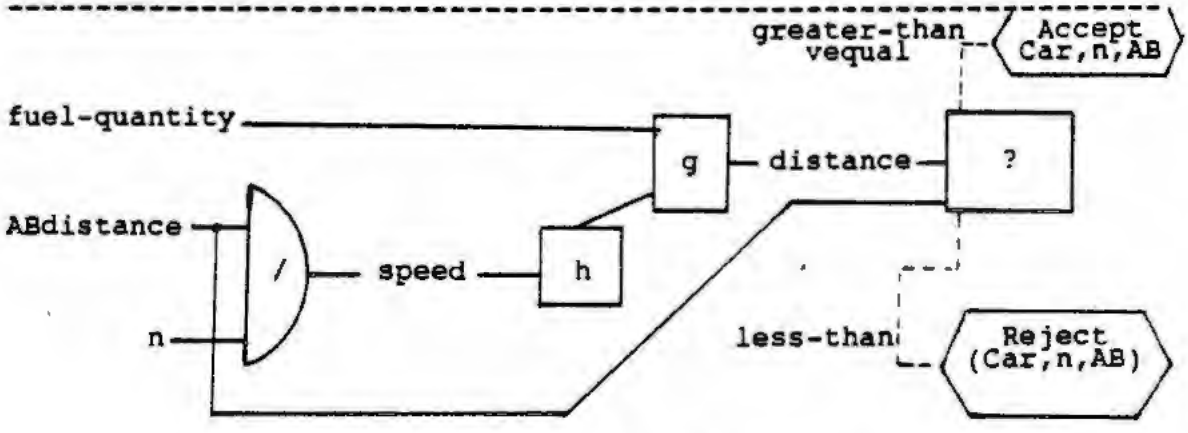
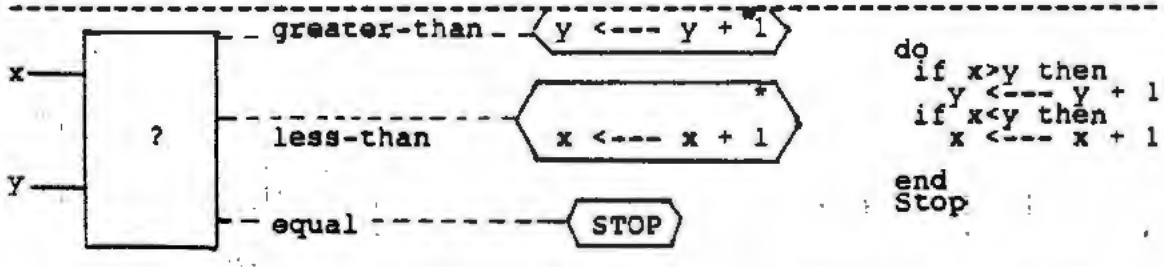


Figure 10: CAN determining wheather car can cover a given distance in a given time

Semantically, the processing of a CAN consists of a usual constraint propagation process (Sussman and Steele 1980) starting from the variables input as arguments to the CAN. Whenever a constraint is considered in the propagation process, attached activities are checked and those which can be executed are executed.

It is allowed that the attached activities modify the variables whose values have previously been propagated. In this case, affected propagations are re-done automatically, as in a constraint propagation system when variables are modified and dependecy records are used to redo affected propagations. Figure 11 shows a simple CAN carrying out the computation shown in its right side. The semantics requiring that affected propagations be redone automatically achieves the effect shown in the right side without requiring explicit control flow links. Note the use of the '*' to specify the activities which can be repeated any number of times (if a number n is placed instead of the '*', the activity can be repeated ar most n times). This has the advantage of specifying computations in a very declarative manner. Also, it is directly interpretable (e.g. using a constraint propagation language) thus satisfying the initial requirement of simulating conceptual models. We are also studying this general manner of expressing computations in constraint propagation networks separatedly as an AI programming language (Barbuceanu 1990).



a) CAN - The star specifies that the activity can be performed any number of times

b) Equivalent algorithm

Figure 11: CAN and equivalent algorithm

3.3.2 Constraint Procedures

The CAN formalism has an implicit reactive nature in that all modifications of variables automatically trigger re-propagations. Sometimes this may be difficult to regulate and a more procedural control mechanism may be needed instead. CP-s provide such a formalism. CP-s specify activities in the usual procedural manner used in conventional languages. However, they also provide a number of constructs which express control by using the constraints declared in the conceptual model. These constructs fall into two classes. Iteration constructs iterate the activities specified in a constraint's cases as long as the constraint is satisfied and a special condition holds. Selection constructs evaluate a constraint and execute the activities associated with the satisfied case. We represent CP-s in the JSP graph notation (Cameron 1986), which provides clear hierachical decompositions. This notation is extended to handle the constrained iterator and constrained selector constructs.

Figure 12 shows the computation expressed in figure 11b as a CP presented in the JSP notation. The '*' in the "?" constraint box specifies iteration any number of times. A number n would impose at most n iterations. At each iteration, the constraint is evaluated on its arguments, the satisfied case is determined and the activity specified for that case is executed.

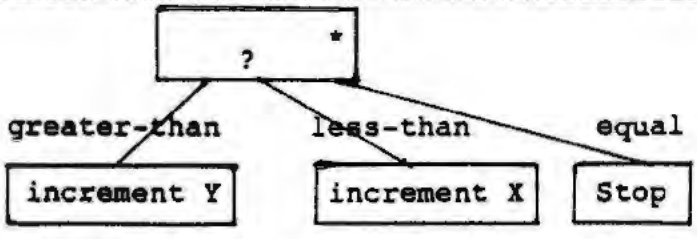


Figure 12: CP for algorithm in fig. 11b

Figure 13 shows a CP performing the activities shown in the CAN given in figure 10. In this CP, the "?" constraint is used in a selector (the '0' denotes this) which evaluates it and executes the activity associated to the satisfied case.

CAN-s and CP-s have an important characteristic in common. They relate the control flow to the cases in which constraints are satisfied. This is useful when maintaining models, as modifications of the constraints (such as removing, creating, refining or merging cases) can be easily propagated over the activities of the model. This property helps solving the problem of automating the maintenance of the procedural part of domain models (see (Balzer 1985) for a discussion of these issues).

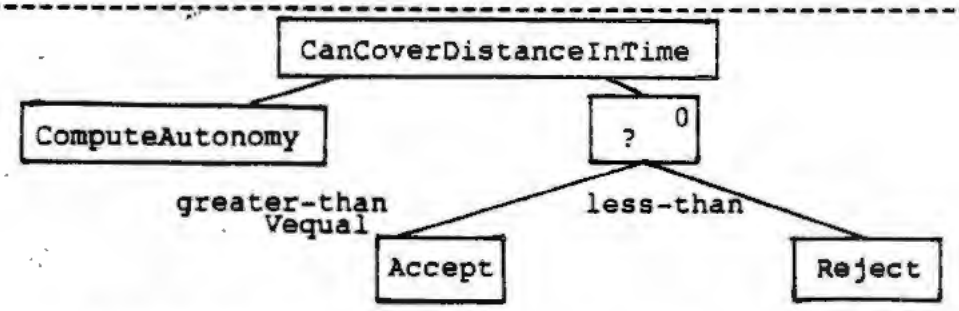


Figure 13: CP equivalent to the CAN in figure 10

4. ABSTRACT KNOWLEDGE PROCESSING MACHINES

In this section we show how AKPM-s can be specified using the CML and how different AKPM-s can be evolved from existing ones. We use as an example a task-specific architecture called Hierachical Refinement. This is a simplified version of a more complex architecture presented in (Barbuceanu, Trausan and Molnar 1989, 1990).

4.1 Description of the task-specific architecture

Hierachical Refinement is a generic problem solving method for synthesis tasks. It views problem solving as a top-down refinement process which takes as input a generic description - i.e. one which applies to many objects in the domain - and produces as output a more refined description - i.e. one which applies to fewer objects in the domain. Hierachical Refinement is thus a process of incremental elimination of the ambiguity associated to partial descriptions.

Assume we are modeling a design process whose purpose is to produce the design of a house. Initially we start with a very general specification of the house as shown in fig. 14. This specification is ambigous in that it lets the features of the house unspecified or loosely constrained. We know only that the Living must be large, the Kitchen well equiped and the Bedroom-s ordinary rooms. This allows very many actual house configurations to stand for this description.

```
[House
  Bedroom [a Room]
  Living [a LargeRoom]
  Kitchen (and [a Room]
               [a WellEquipedKitchen])]
```

Figure 14: Initial House description for Hierachical Refinement

Refinement reduces this incompletness by further completing and constraining the specification. Thus, after several refinement phases the description from fig. 14 might look like in fig. 15. Here, the roles of the initial description have been refined to more specific descriptions. The set of house configurations consistent with the description in fig. 15 has been reduced to a proper subset of the initial one. Refinement continues in this manner, reducing the size of the set of allowed interpretations, until a satisfactory degree of specificity is attained.

```
[House
  Bedroom ([a Room Surface 16]
           [a Room Paint Orange])
  Living [a Room Surface 24
                Paint Pink]
  Kitchen (and [a Room Surface 12]
               [a WellEquipedKitchen])]
```

Figure 15: Refined House description

To refine descriptions, Hierarchical Refinement supports several refinement operators which, applied on a given description, yield a more refined one. In this brief presentation we mention only three of them, Expand, Anchor and AsIs. The Expand operator refines a concept by taking each of its roles and recursively refining their fillers. The Anchor operator returns a refined form of the given concept, which will not be further refined. AsIs simply leaves the description as is.

Refinement takes place in a common working space (the Blackboard) where several concepts can be symultaneously refined. Each concept or description is assigned a refinement task in an agenda. The main loop of the refinement process consists of the following three steps applied until a termination condition becomes true:

1. Select the next task (concept or description) to be refined.
2. Select the refinement operators to be applied.
3. Apply the operators.

For example, consider the concept in figure 14. Applying the Expand operator on this concept results in creating three new tasks in the agenda, one for the filler of the Bedroom role, one for the filler of the Living role and another for the filler of the Kitchen role. At the next cycle, the [a Room] concept (filler of Bedroom role) is selected. Applying Expand again produces new tasks for the roles of this concept e.g. Surface, Paint, etc. When the task for the Living role is selected, the Anchor operator may be used to execute it. Anchor-ing may retriere from a data base a completely refined Room description suitable for filling this role, as in fig. 15.

This problem solving model combines constructive activities (the Expand operator) with possibilities for search (the Anchor operator). In (Barbuceanu, Trausan and Molnar 1989) it is shown that it can be used to simulate some of the recently defined problem solving types such as hierarchical diagnosis and hierarchical design (Chandrasekaram 1987). We have extensively used hierachical refinement in design applications (Barbuceanu 1985a, 1985b).

4.2. The Hierachical Refinement AKPM

Let us now specify the inputs, outpus and internal mechanism of the AKPM implementing Hierachic Refinement.

First, to refine descriptions in the Hierachical Refinement manner we need a way of associating specifications of the refinement process with the concepts or structured descriptions to be refined. One obvious manner of doing this is to introduce roles for these specifications. These roles will be interpreted by the problem solver when carrying out refinement. Acquisition of values for them will be assisted by the KA tool.

Here is a possible organization of these roles. To specify the manner of refining a concept, the RefineBy role is used. Its value may be one of expand, anchor or asis. An additional ToAnchor role supplies the anchoring operator. To decide wheather a concept can be refined, role StartIf holds a predicate which will take this decision at run-time. It is also possible to specify side effect activities to be performed before or after refining a description. This leads us to the form of the input and output concepts of the AKPM. These concepts are shown as the RefinableConcept-s in the terminology in fig. 16.

Domain concepts will be acquired and placed in this taxonomy as needed - an example is the House concept in figure 16 which imports a number of refinement specifications. The terminology in

figure 16 also defines the terminology specific to the internal mechanism of the AKPM. This mechanism is actually the model of the arhitecture's inference engine. This includes the RefinementSystem which holds the agenda and the RefinementTask which holds attributes of the refinement tasks.

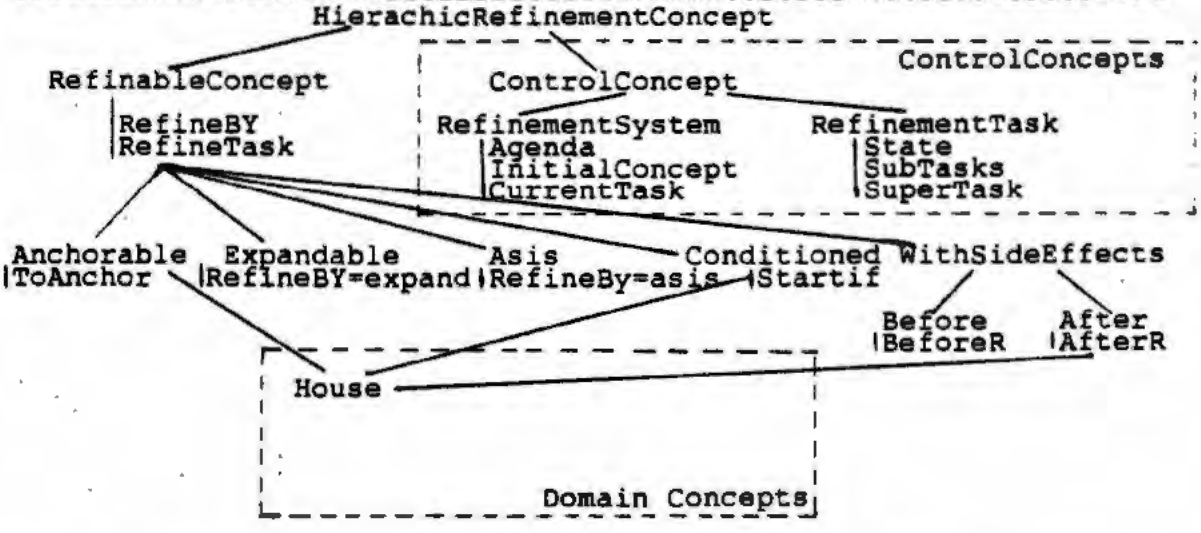
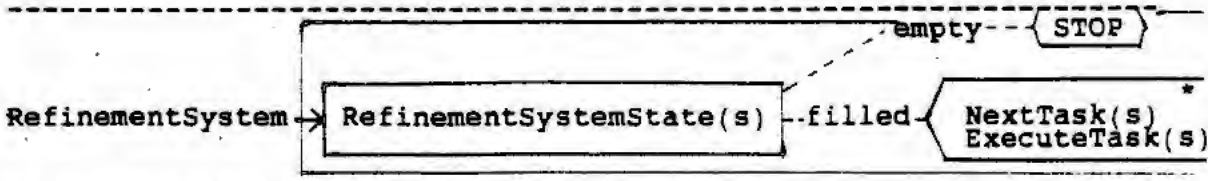


Figure 16: Terminology for the Hierarchical Refinement problem solving system

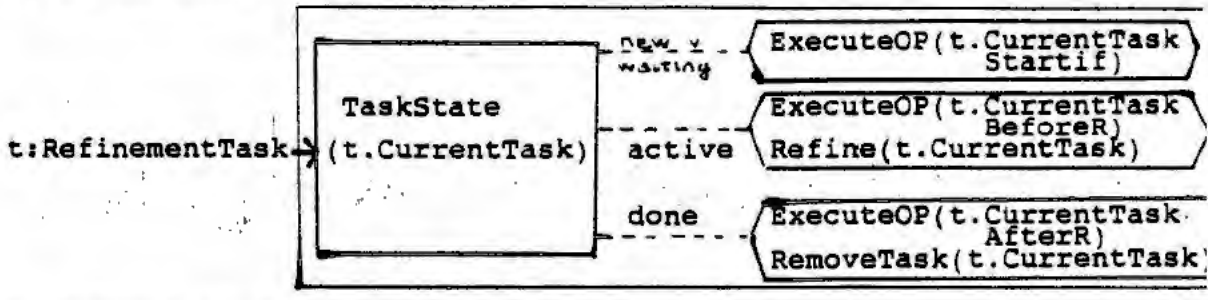
Figure 17 shows some constraints on the concepts in figure 16. They refer to the possible refinement modes, the possible task states and the states of the refinement systems. Figure 18 shows three CAN-s describing the procedures by which the inference engine will carry out the refinement process. These procedures include the top level loop of the system and the procedures for selecting and executing refinement tasks.

```
RefinementMode
 arguments
  x:RefinableConcept.RefineBy
 cases
 expand (x = expand)
 anchor (x = anchor)
 as-is (x = as-is)

TaskState
 arguments
   x:RefinementTask.State
   y:RefinementTask
 cases
 new (x = new)
 active (x = active)
 waiting (x =waiting)
 done (or (x = done)
          (and (x = active)
               (if (has-subtasks y)(all-subtasks-done y))))

RefinementSystemState
 arguments
   x:RefinementSystem.Agenda
 cases
   filled (not (null x))
   empty (null x)
```

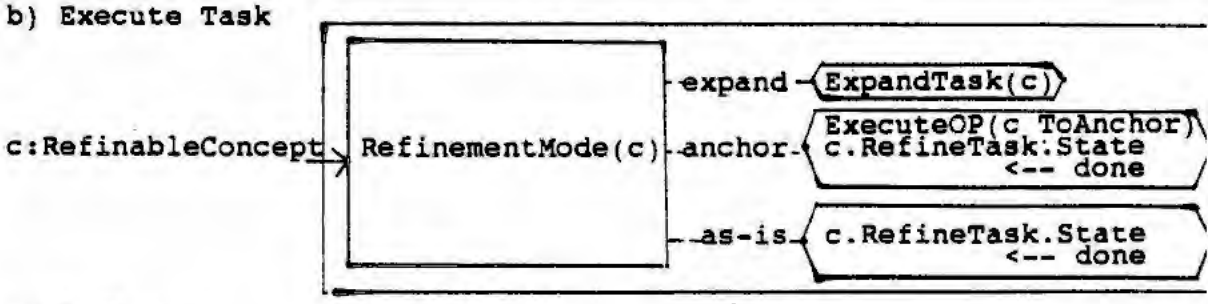
Figure 17: Constraints for Hierarchical Refinement



a) Hierarchical Refinement Cycle



b) Execute Task



c) Refine

Figure 18: Some CAN-3 for Hierarchical Refinement

Finally, the overall description of the Hierarchical Refinement AKPM can be given using the CML concept language as in fig.19.

```
[HierarchicalRefinementAKPM
  Inputs [a RefinableConcept]
  Outputs [a RefinableConcept]
  Mechanism [a Mechanism
              InternalConcepts [a ControlConcept]
              Activities (HierarchicalRefinementCycle
                          ExecuteTask Refine...)
              InitialActivity HierachicalRefinementCycle]]
```

Figure 19: The Hierarchical Refinement AKPM

4.3 Evolving new AKPM-s from Hierarchical Refinement

Hierarchical Refinement is a general hierarchic problem solving model. Recent research has produced more specific hierarchical problem soving models such as the hierarchic diagnosis and hierarchic design generic tasks (Chandrasekaran 1987). As shown in (Barbuceanu, Trausan and Molnar 1989) it is easy to simulate these with hierarchical refinement. A better solution however, motivated by reasons of epistemological adequacy and computational efficiency, is to modify hierarchical refinement to produce such problem solving systems. One of the major goals of our approach is to support this kind of evolution which will lead to having families of more or less related problem solving systems integrated in the same programming environment. In this section we show how these two problem solving models can be obtained by modifying hierarchic refinement and discuss the degree in which such an evolution can be automated.

4.3.1 Hierarchical diagnosis

Hierarchical diagnosis (Chandrasekaran 1986, 1987) is a classificatory task performed by a top down strategy known as "establish-refine". Each concept involved in the process represents a more or less general malfunction hypothesis. In the problem solving process, each concept tries to establish or reject itself. The knowledge pertinent to establishing or rejecting a concept is locally contained in the concept. If a concept is established, control is passed to its sub-concepts which will try to establish or reject themselves a.s.o. until a tip concept is reached or no concept can refine any further. If a concept is rejected, the entire branch of the hierarchy under it will be pruned.

Of course, this is close to Hierarchical Refinement and can be easily simulated with it. To obtain better epistemological adequacy and computational efficiency, we prefer to modify hierarchical refinement and turn it into this form of problem solving. Here are the modifications to be performed.

First, we modify the conceptual vocabulary in the following manner:

1. Transform the StartIf operator (which was used to decide wheather a refinement task is to be executed) into two operators, Establish and Reject, deciding when the concept must be refined or respectively rejected.
2. Adopt refinement by expansion as the single refinement option. This makes RefineBy operator useless. It is eliminated together with concepts Anchorable and Asis.
3. WithSideEffects concepts do not play an important role and may be eliminated too.

These modifications lead to reorganizing the conceptual vocabulary taxonomy. The new taxonomy may look as in figure 20.

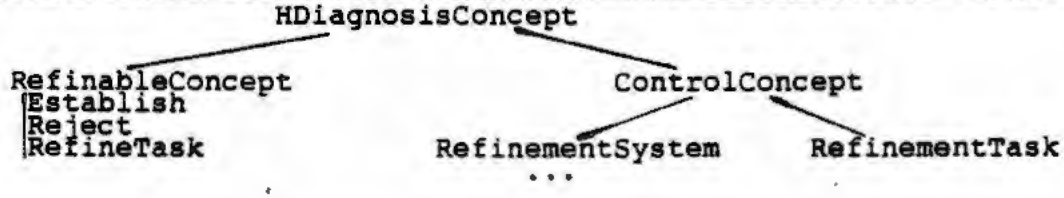


Figure 20. New terminology for hierarchical diagnosis

Second, constraints and CAN-s are modified in consequence. The RefinementMode constraint will dissapear, being replaced in CAN-s with the activities associated to the expand case which is now implicit. As a consequence, the Refine CAN in figure 18-c will be replaced by the ExpandTask activity (e.g. in the ExecuteTask CAN). The ExecuteTask CAN must be modified to account for the replacement of the StartIf operator with the new operators Establish and Reject.

The explicit representation of modeling assumptions in the conceptual model allows to mechanically support many of the above modifications. Such support may be given for propagating the effects of modifying the terminology upon constraints and CAN-s which make explicit use of the terminology. Thus, replacing a role with others, removing concepts or roles, eliminating constraints by freezing one of their cases are all examples of modifications whose execution can be mechanically supported.

This sort of support has been investigated in (Balzer 1985) for similar frame based models. This research has introduced a classification of modifications which can be supported. In the context of our conceptual modeling language the classes of such modifications are more numerous as shown in previous investigations (Barbuceanu and Trausan 1990).

4.3.2 Hierarchical design

The hierarchical design model (Chanchasekaran 1987, 1987) consists of a similar hierarchy of concepts called design specialists. Specialists higher up in the hierarchy deal with

more general aspects of the system being designed while those lower in the hierarchy design more specific portions of the system. Each specialist contains locally the knowledge required to accomplish its part of the design task. A specialist contains several explicit design plans which are alternative ways for completing its task. A specialist also contains a plan selector for selecting at run-time the most suitable design plan from among the existing ones. Each design plan in its turn has a so called design sponsor which decides on the appropriateness of the plan. Plan selectors evaluate the judgement of plan sponsors to choose a design plan.

In this model, control proceeds from the top-most specialist to the lowest. Each specialist selects a design plan which is then executed. Design plans may refer sub-specialists, in which case control is passed to those sub-specialists. It is also possible to handle plan failures and to control redesign activities suggested by these failures.

Again, this can be simulated with the Hierarchical Refinement machinery, but for clarity of modeling and computational efficiency we may try to adapt hierarchic refinement to explicitly model this form of hierarchical design.

Concerning the conceptual vocabulary, instead of the various RefinableConcept-s we now have two kinds of concepts, one for design specialists and the other for design plans. Specialists have roles for the available design plans (filled with DesignPlan concepts), the plan selector operator and a redesign operator. DesignPlan-s have roles for the plan sponsors and a success criterion for the plan. Figure 21 shows this organization.



Figure 21: Terminology for hierarchical design

This requires two different CAN-s to replace the initial Refine CAN from figure 18-c. For DesignSpecialist-s, the new Refine CAN must apply the design sponsors for each design plan and then select a plan with the plan selector. For DesignPlan-s, the Refine CAN will execute the plan with a special operator, test success with the SuccessCriterion and if failed invoke the RedesignOp-erator to fix the problem.

Notice that as before, ControlConcept-s from the original Hierarchical Refinement model are reused in this model.

The two examples presented in this section illustrate a number of advantages of the approach:

1. Significant parts of the models can be reused from one task to another.
2. Due to the explicit representation of modeling assumptions, the modification process can be guided and supported by mechanized tools which are specific to the conceptual modeling aproach and not to the various tasks modeled.
3. A knowledge engineering environment based on this approach can contain many different problem solving systems, more or less related. Users can select the most appropriate tool for their needs or proceed to build their own when none is good enough. Building such tools is supported by the degree of sharing among models and tools which can be exploited if one has a unique conceptual modeling language as the foundation of the environment.

5. MODELING

After presenting the CML and the AKPM notion, we now discuss the support provided for the modeling step of the approach.

As other authors (Klinker 89, Chandrasekaran 1990) we assume that the users of the environment - although non-programmers - can specify the inputs and outputs of the knowledge based system they wish to build and the knowledge available for solving the problem. In this case a user controlled backward reasoning process starting from the outputs of the desired system can be used to identify the configuration of AKPM-s which would solve the user's task.

Let us illustrate it with a diagnosis task example given in (Chandrasekaran 1989). The output of the system should be a set of hypotheses which best explains the set of input observations. Given that the MODELS knowledge base contains both a general conceptual taxonomy and a problem type taxonomy as a shared background for communication and that existing AKPM-s are indexed in a way which relates them to these taxonomies, the environment may retrieve the Abductive Assemply AKPM as capable of producing the desired output. This AKPM requires however as input a set of plausible hypotheses. There may exist several AKPM-s producing these hypotheses, but differring in the kinds of knowledge needed and methods used. E.g. using Bayesian inference requires apriori and conditional probabilities which may or may not be available. The appropriate AKPM should hence be selected according to the availability of the required knowledge and any other relevant criteria (e.g. performance). If the Hierarchical Classification AKPM is chosen, it will require as inputs the initial hypotheses to be confirmed or rejected. These may come from a Hypothesis Matching AKPM which heuristically matches observations with possible hypotheses. The observations needed by Hypothesis Matching must be supplied at a higher level of abstraction than that of the input data. This introduces the last AKPM, a Data Abstraction one which works directly on the inputs available to the desired system (see also (Clancey 1985)) producing the abstracted data required.

When determining the AKPM-s it is useful to determine their abstraction levels as well - in the KADS sense (Breuker and Wielinga 1987). Abductive Assembly and Hierarchical Classification are at the task level, Hypothesis Matching is at the inference level and Data Abstraction at the domain level. There is no strategic level AKPM. A strategic AKPM is however need at least for playing the role of the "main program", in this case having the job of calling the four AKPM-s in sequence. More careful analysis shows that the two task level AKPM-s, Abductive Assembly and Hierarchical Classification interact in a more complex manner. When explanations can not be assembled, other hypotheses may be needed and hence should be established by Hierarchical Classification. This points to the need for yet another strategic AKPM managing this dialoque. Thus we are led to introducing two new machines and to the final configuration of the resulting AKPM shown in fig. 22 in terms of the JSP notation.

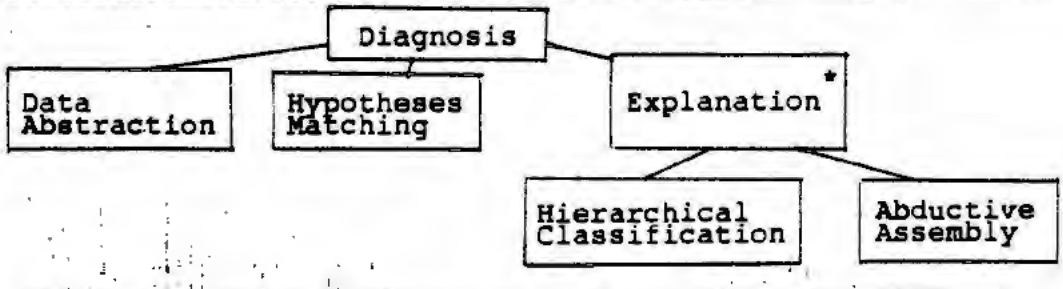


Figure 22: Final configuration of the Diagnosis AKPM

6. GENERATION

Generation is the second step of the MODELS approach. It translates the conceptual model into an executable target AI language. The Generation step is divided into two phases. The Specification phase produces the specifications of the knowledge based system, adding the detail information needed for specifying the AKPM viewed as a piece of software, but not needed in the conceptual model. The Compilation phase compiles the knowledge based system in the target language.

6.1 Specification

The output of this phase is a description of the following elements:

1. The representation structures in which the knowledge base will be encoded;
2. The knowledge acquisition tool which will elicit the content of the knowledge base;
3. The problem solving mechanism (inference engine) which will process knowledge to solve problems.

These descriptions must contain all information required to generate the actual system in the target language.

Our assumption is that the target language is a state-of-the-art frame language with inheritance, demons, data-types, methods and eventually some provisions for integrating rule programming (examples of such languages are KEE (Fikes and Kehler 1985), SRL (Wright and Fox 1984), LOOPS (Bobrow and Stefik 1981) etc.).

6.1.1 Specifying representation structures

This phase determines the following three aspects:

1. The concepts from the conceptual model which will be represented as generic frames in the knowledge base.
2. How these frames share structure through inheritance.
3. How constraints will be enforced in the actual knowledge base.

Concerning the first issue, it is neither required nor recommendable to have a frame for each concept in the conceptual model. Figure 23 shows how the conceptual taxonomy from figure 5 can be encoded in terms of fewer generic frames. These frames will have slots for each role of the concept, cardinality restrictions and data types derived from the corresponding concepts. Figure 23 also shows inheritance links among frames. These must not be confused with subsumption relations. The latter have a conceptual nature being induced by a subset-superset relation between concepts extensions, while the former reflect sharing of structure between frames.

The conceptual model also contains constrains among concepts and roles. These must be maintained and effectively computed at the level of the frames as constraints among frames and slots. The problem is that if constraints are arbitrary predicates it may not be always possible to determine candidate values for all variables. This has the effect of forbidding some propagations. To specify which propagations are effectively allowed and which are not we use the CAN formalism and the meta-predicates known and unknown as illustrated in figure 24 where it is not possible to determine the full-quantity when distance and speed are known.

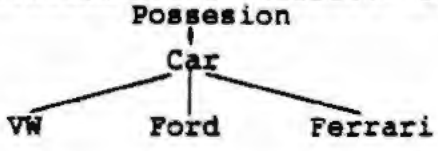


Figura 23. Selected frames with inheritance from conceptual taxonomy

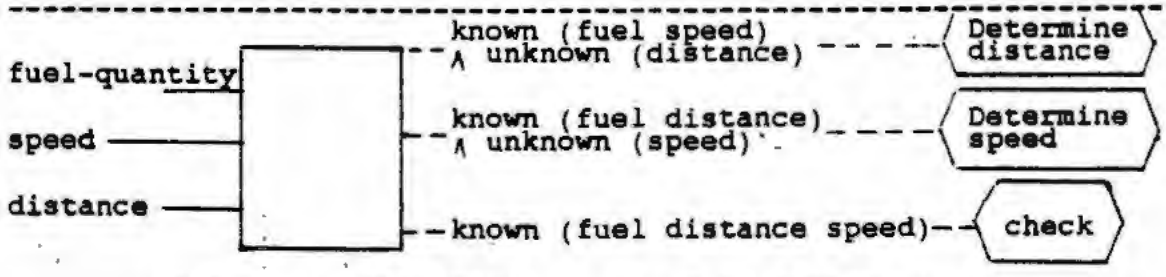


Figure 24. CAN specifying allowed propagations

6.1.2 Specifying the knowledge acquisition tool

The purpose of the KA tool is to elicit instances of the generic frames determined in the previous phase. In the domain of figure 23 for example, this consists of eliciting various Car instances which will be integrated in the frame network. Because the structure of generic frames is known - that is the slots, their cardinality, data types and constraints amongst them - the tool will know what to ask for, in what range the answers must lie and what checks to perform upon entered knowledge (Gruber and Cohen 1987).

During the specification of the KA tool phase information is added concerning:

1. The nature of the slot values to be acquired;
2. The graphical forms in which knowledge elicitation will take place.

As in PROTEGE (Musen 1989) we recognize three kinds of values. The first are values in the generic frames which can not be modified in instances (e.g. the manufacturer of Ford-s is allways "Ford"). The second are values which are set at knowledge elicitation time and can not be altered dynamically (e.g. the power of an engine). The third are values which are set and modified dynamically during problem solving (e.g. the speed of a car).

These distinctions induce a number of similar distinctions on constraints. Constraints among values exclusively of the first kind must be satisfied prior to knowledge elicitation. Those among values of the second kind are satisfied at knowledge elicitation time and the KA tool may use such constraints to infer values. Constraints among values of different kinds necessarily restrict the possible propagations (e.g. a value of the first or second kind can not be modified during problem solving to satisfy a constraint, but those of the third kind can).

6.1.3 Specifying the inference engine

The inference engine follows from the activities described within the conceptual model through CAN-s or CP-s. These are complete descriptions of computations, but for a useable program information must be also specified relative to:

1. Error handling activities;
2. Input/output activities.

Error handling activities are specified as activities associated with error cases in constraints. Such activities may require imposing certain control flows such as leaving the CAN (for fatal errors) or resuming it (for corrected ones).

For variables which must be obtained and/or displayed interactively, input/output activities must be added which will specify data formats, validity checks, etc.

6.2 Compilation

This produces the three components of the problem solving system in the target language.

First, the initial content of the knowledge base - corresponding to the generic concepts in terms of which domain knowledge will be elicited - must be translated into the target language.

Second, the KA tool can be generated in at least two basic ways. One is to have a single meta-interpreter for specifications (which appear as annotated conceptual models) which would work for any domain. Another is to compile a specific KA tool for each conceptual model.

Third, as concerns the inference engine it must be generated from CAN-s or CP-s which describe knowledge processing in the domain. A similar choice exists for its implementation. It can be implemented by a domain independent CAN or CP interpreter or by domain dependent programs compiled for each model. The latter

solution seems to be required for production use while the former may be employed as a rapid prototyping method. There exist results on compiling constraint network into rules, which can be used for this purpose (Subramanian and Freuder 1989).

7. SUMMARY AND CONCLUSIONS

We have presented the design of an environment for knowledge acquisition and expert system construction based on the task-specific architecture school of thinking. The big advantage of task-specific architectures is that they ensure both epistemological adecquacy and computational efficiency. To support this approach requires shifting the focus of attention from programming as coding to programming as model composition and from knowledge acquisition as expertise transfer to knowledge acquisition as model construction (Clancey 1989).

We have proposed the following solutions to operationalize the task-specific architecture approach.

1. A formal conceptual modeling language as the unified base for describing the various schemes of knowledge representation and processing that task-specific architectures rely on. This language is novel in integrating structural description languages with formal subsumption semantics, with constraints and other novel mechanisms for describing activity. The language is executable allowing knowledge level prototyping.

2. A formalization of task-specific architectures as Abstract Knowledge Processing Machines whose inputs, outputs and internal mechanism are uniformly described in CML. AKPM-s can be built by integration, evolution or creation. They are directly simulable, but can also be compiled into more efficient code.

3. A two stage method for producing knowledge based systems: conceptual modeling, where one builds a formal conceptual model of the knowledge based system as an AKPM and generation, where one generates the target language form of the system. Generated systems comprise the knowledge acquisition tool, the inference engine and the initial knowledge base.

4. Generation of the knowledge acquisition tool follows by meta-programming from the conceptual model.

5. Evolution and maintenance are supported by propagating the consequences of modifying conceptual models upon specifications and implementations. These are also supported by the high degree of sharing among models which occurs in this environment.

We have illustrated the proposal with examples concerning both application level modeling and more complex task specific architecture.

The approach can be implemented with state-of-the-art technology. Attention must be payed at implementation to aspects such as the subsumption procedure, the compilation of CAN-s into rules and the implementation of the modification consequence propagation mechanisms.

Finally, we believe this kind of system could make a step towards a common knowledge engineering and software engineering foundation based on domain modeling. It supports a modeling view on programming which we hold since a longer time (Barbuceanu 1986). According to this view programming is a process in which one first designs a task specific modeling language which "vividly" (Levesque 1986) reflects the patterns of information processing in the domain and then applies this language in an execution environment.

REFERENCES

1. Alexander J.H., Freiling M.J., Shuman S.J., Rehfuss S., Messik S.L. (1986) Ontological Analysis: An Ongoing Experiment, Proceedings AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems, Banff, Canada 1986.

2. Balzer R. (1985) Automated Enhancement of Knowledge Representation Languages, Proceedings of Ninth International Joint Conference on Artificial Intelligence, Los Angeles, 1985.

3. Barbuceanu M. (1985a), An Object Centered Framework for Expert Systems in CAD, In: I.S. Gero (ed) Knowledge Engineering in Computer Aided Design, North Holland, 1985, 223-253.

4. Barbuceanu M. (1985b), A Domain Independent Knowledge Engineering Architecture for Design, Proceedings of Fifth International Worksop on Expert systems and Their Applications, Avignon, France 1985.

5. Barbuceanu M. 1986, Knowledge Based Development of Reusable and Evolutionary Software, Proceedings of the Sixth International Workshop on Expert Systems and Their Applications, Avignon, France 1986.

6. Barbuceanu M., Trausan-Matu Stefan, and Molnar B. (1989) Concurrent Refinement of Structured Objects: A Language for Declarative Knowledge Programming Based on Specifications and Annotations, Proceedings of Expert Systems '89, Brighton, UK, 1989.

7. Barbuceanu M., Molnar b., Trausan-Matu Stefan S. (1990), Concurrent Refinement: A Model and Shell for Hierarchic Problem Solving, Proceedings of Tenth International Workshop on Expert Systems and Their Applications, Avignon, France 1990.

8. Barbuceanu M. (1990), Enhancing Object Oriented Modelling with Constraints, forthcoming 1990.

9. Barbuceanu M., Trausan-Matu S. (1990), MODELS: Towards a Language Construction Approach to Expert System Devolopment, to appear in "Studies and Researches in Computers and Informatics" 1990, Bucharest.

10. Bobrow D.G., Stefik M.J (1981). The LOOPS Manual TR-KB-VLSI-81-13, Xerox Palo Alto Research Center, Palo Alto, CA, 1981.

11. Brachman R.J., Gilbert V.P., Levesque H.J. (1985), An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON, Proceedings of Ninth International Joint Conference on Artificial Intelligence, Los Angeles, 1985.

12. Brachman R.J., Schmolze J.G (1985). An Overview of the KLONE Knowledge Representation System, Cognitive Science 9(2) 1985.

13. Brachman R.J., Levesque H.J. (1984), The Tractability of Subsumption in Frame-Based Description Languages, Proceedings of the American Association for Artificial Intelligence, Austin, Texas 1984.

14. Breuker, J. and Wielinga, B., (1987) Use of Models in the Interpretation of Verbal Data. In Knowledge Acquisition for Expert Systems: A Practical Handbook, Kidd, A.L. (ed) New York, Plenum.

15. Cameron, J.R., (1986) An Overview of JSD, IEEE Transactions on Software Engineering, feb. 1986, 222-240.

16. Chandrasekaran B., (1986) Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert Systems Design, IEEE Expert Fall 1986.

17. Chandrasekaran B., (1987) Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks, Proceedings of Tenth International Joint Conference on Artificial Intelligence, Milan Italy 1987.

18. Chandrasekaran, B. (1990) Task Structures: New Directions in Generic Tasks, Invited lecture at the 10-th Intl Workshop on Expert Systems and Their Applications, Avignon 1990.

19. Clancey, W. (1989) The Knowledge Level Reinterpreted: Modeling How Systems Interact, Machine Learning 4, 3/4 dec. 1989, 285-293.

20. Eshelman L., McDermott J. (1986) MOLE: A Knowledge Acquisition Tool That Uses Its Head, Proceedings of the American Association for Artificial Intelligence, Philadelphia 1986.

21. Fickes R., Kehler T. (1985) The Role of Frame Based Representations in Reasoning, Communications of the ACM, 28, 1985.

22. Gruber T., Cohen T. (1987) Principles of Design for Knowledge Acquisition, Proceedings of IEEE Conference on Artificial Intelligence Applications, 1987.

23. Herman D., Josephson I., Hartung R., (1986) Use of DSPL for the Design of a Mission Planning Assistant, Proceedings of IEEE Conference Artificial Intelligence in Government, 1986.

24. Klinker, G. (1988) KNACK: Sample Driven Knowledge Acquisition for Reporting Systems. In S. Marcus (ed): Automating Knowledge Acquisition for Expert Systems, Boston, Kluwer Academic Publishers.

25. Klinker G., (1989) A Framework for Knowledge Acquisition, Proceedings of Third European Workshop on Knowledge Acquisition for Knowledge Based Systems, Paris 1989.

26. Levesque H., (1986) Making Believers Out of Computers, Artificial Intelligence 30(1) 1986.

27. Luck K., Nebel B., Peltason C., Schmeidel A., (1987) The Anatomy of the Back System, KIT Report 41, Fachbereich Informatik, Technische Universitat Berlin, Berlin (West) 1987.

28. Marcus S., McDermott J., Tianran W., (1985) Knowledge Acquisition for Constructive Systems, Proceedings of Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, M85.

29. MacGregor R., Bates R., (1987) The Loom Knowledge Representation Language; ISI IRS-87-188, USC/ISI Marina del Rey, CA, 1987.

30. McDermott J.(1988) Preliminary Steps Towards a Taxonomy of Problem Solving Methods, S. Marcus (ed): Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers, 1988.

31. Musen M.A., (1989) Knowledge Acquisition at the Meta Level: Creation of Custom-Tailored Knowledge Acquisition Tools, SIGART Newsletter 108, 1989.

32. Nebel B., (1988) Computational Complexity of Terminological Reasoning in BACK, Artificial Intelligence 34, 1988.

33. Patel-Schneider P.F., (1984) Small Can be Beautiful in Knowledge Representation, Proceedings IEEE Workshop on Principles of Knowledge Based Systems, Denver Colorado, 1984.

34. Schmolze J.G., (1985) The Language and Semantics of NIKL, BBN Inc., Cambridge MA, 1985.

35. Shum S.K., Davis J.F., (1986) Application of CSRL to Malfunction Diagnosis in Complex Processing Plants, Proceedings of IEEE Conference on Artificial Intelligence in Government, 1986.

36. Subramanian S., (1989) Freuder E.C., Compiling rules from constraint satisfaction problem solving, SIGART Newsletter No 108, 1989.

37. Sussman G.J., Steele G.L. (1980) Constraints: A Language for Expressing Almost Hierarchical Descriptions, Artificial Intelligence 14,1, 1980.

38. Twine, S., (1989) A Model for the Knowledge Acquisition Process, Proceedings of Third European Workshop on Knowledge Acquisition for Knowledge Based Systems J. Boose, B. Gaines, J.G. Ganascia (eds) Paris, 1989.

39. Wright J.M., Fox M.S., (1984) SRL 15 User Manual, Tech. Rept. Carnegie Mellon University, Robotics Institute 1984.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Acquiring and Refining Indices According to Context

Guy A. Boy[1]
Artificial Intelligence Research Branch
NASA-Ames Research Center
Mail Stop 244-17
Moffett Field, CA 94035, USA

(415) 604-3369
Email: boy@pluto.arc.nasa.gov

February 1990

Abstract

This paper presents a new approach to index acquisition that utilizes existing indices and modifies them by introducing the concept of abnormal conditions. It is based on a *block* representation that relates preconditions (contextual and triggering) to postconditions (goals and abnormal conditions) through the selection of actions. Our system utilizes on-line user requirements and suggestions to either reinforce current actions in case of success or to generate new recovery actions in case of failure. This allows an intelligent assistant system to provide helpful responses, even when no robust user model is available. This paper also presents a mechanism for incrementally constructing contextual conditions. These contextual conditions generate a better structure of the block knowledge base, presenting only the actions that are most likely to be useful, and thus limiting the problem of selection. (The user can, however, access all the indices at any time.) This model consequently improves the performance of the system.

We have applied this approach to the problem of Computer Integrated Documentation (CID), partially automating management and maintenance of the knowledge about an engineered system.

Paper to be included in the Proceedings of the Fifth AAAI-Sponsored
Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November 4-9, 1990

---

[1] Dr. Boy is currently working at NASA under contract no. NAS2-11555 to Sterling Federal Systems.

1 Introduction

When a beginner starts to operate a particular system, he starts learning formal procedures which he will improve incrementally over time. The procedural knowledge improves augmenting his knowledge base, and by transforming various entities of available procedures into more contextual procedures that are better suited for routine use. Results of this transfer lead incrementally to more operational knowledge. Such a knowledge compilation mechanism is similar to (Anderson, 1983; Rosenbloom, 1983). The main claim of our approach is that contextual knowledge can be gained through experimentation. Thus, computers can provide experimental support for capturing contextual knowledge. In this research we consider document indexing as procedural, i.e. indices are particular procedures that are modified incrementally by experimentation.

This paper makes two contributions. First, we present a new approach to index acquisition that utilizes existing indices and modifies them by introducing the concept of abnormal conditions. It is based on a *block* representation. Its advantage in acquiring knowledge is its naturalness for humans - the resulting knowledge bases are similar in form to technical and operation manuals. Furthermore, the block representation is very convenient for incrementally building operational knowledge. Our system utilizes on-line user requirements and suggestions to either reinforce current indices in case of success or to generate new recovery actions in case of failure. This allows the system to provide helpful responses, even when no robust user model is available.

Second, we propose a mechanism for incrementally constructing contextual conditions from scratch. These contextual conditions generate a better structure of the block knowledge base, presenting only the actions that are most likely to be useful, and thus limiting the problem of selection. (The user can, however, access all the indices at any time). This model consequently improves the performance of the system.

In the balance of the paper, we discuss the technique and the implemented system. Section 2 presents an overview of the representation and the technique. Section 3 introduces an application: Computer Integrated Documentation (CID). Section 4 describes the system IARC (*Index Acquisition and Refinement according to Context*), in terms of inputs, output, and processes of each modules. In particular, we will show how this representation is suitable for capturing more appropriate indices in a CID. Sections 5 discusses related work. We conclude in section 6 with limitations and some future research issues.

2 Overview

2.1 Why And How Indices Are Built

Indexing is useful in operational environments. In other words, indexing capabilities are built because they are necessary to retrieve the right piece of information at the right time for accomplishing routine and abnormal tasks. It is generally easier and faster to use currently available search strategies (based on indices) than to re-build them from scratch. As a step towards assisting index creation and maintenance, we examine a specialized class of indices, those that are used in the control and understanding of an engineered system. Generally, people build indices by modifying existing indices according to their own experience. Available written documentation systems are annotated, revised, and re-written. This mechanism is illustrate in Figure 1.

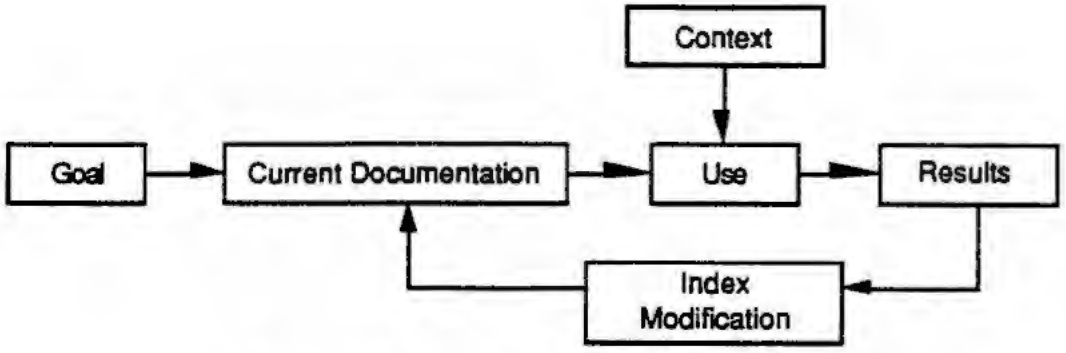


Figure 1. Indexing construction mechanism

2.2 How Indices Are Represented

An index is represented as a *block* containing a description of a set of preconditions (triggering preconditions and contextual conditions), a set of postconditions (goals and abnormal conditions), and a set of actions to achieve the goals (Figure 2).

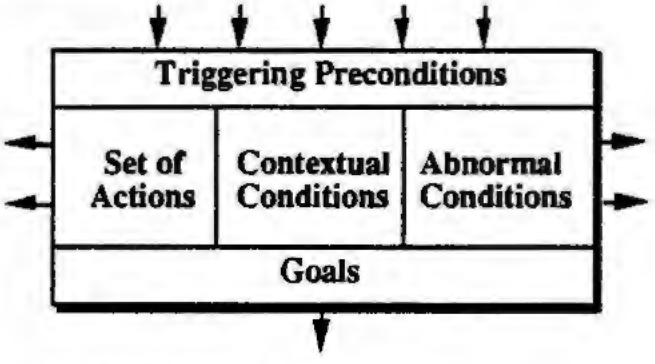


Figure 2. Block representation

Generally, the selection of one (or several) action(s) in a *set of actions* is task-driven. The application of an action is supposed to reach a given sub-set of atomic goals, and can be attached to particular set of abnormal conditions.

Triggering preconditions are conditions which lead to the activation of a set of actions. The triggering preconditions PC(B) of a knowledge block B are represented as a conjunction of predicates. The triggering preconditions of a block can be totally satisfied, partially satisfied, or not satisfied.

Goals G(B) of a block B are represented as a disjunctive set of predicates. Each goal is associated with an action and represents the expected result of this action. Goals of a block can be suggested, not suggested, reached, or not reached. A goal can be suggested by another block, or internally, when the triggering preconditions of its block are satisfied. Otherwise, it is not suggested.

An *abnormal condition* is represented as a disjunction of atomic predicates. The abnormal conditions of a block can be satisfied or not satisfied. When a set of actions is no longer applicable to the current situation, the situation is said to be "abnormal" for this block and an abnormal condition corresponding to this situation must be attached to the block. Abnormal conditions can be associated with the entire block or with a specific set of actions. For instance, during the execution of a navigation procedure in an aircraft (i.e. a particular piece of documentation), a cabin

depressurization in flight is an abnormal condition which leads to the application of a recovery procedure.

Contextual conditions are represented as a disjunctive set of predicates. Contextual conditions in which a block holds define a range for the state of the environment in which the block can be activated and executed. For instance, contextual conditions can be a set of conditions characterizing a failure of a particular device, the current goal of the user, or a mixture of several physical and intentional conditions. Contextual conditions or *context* have been distinguished from the triggering preconditions to express a different grain of representation of knowledge in the hierarchy of triggering preconditions.

For instance, if somebody is solving a problem in a given environment, there are a lot of tacit preconditions which are "obvious" (e.g., constant) in this environment. These preconditions are included in the contextual conditions. Given this definition, if a system is in a context, reasoning can be done on the set of blocks belonging to that context. Thus, triggering preconditions can be made much simpler, leading to faster pattern matching. Contexts are organized in hierarchies. This facilitates the organization of the block knowledge base. For example, in aviation, a flight is generally organized into phases, which are decomposed into sub-phases and finally each sub-phase is described in the form of knowledge blocks that are documented in the form of procedures (Figure 3). In the context of a flight, there are several sub-contexts (before take-off, take-off, etc.). In each terminal context (e.g., take-off), a set of blocks has to be performed. If everything is "normal", the pilot accelerates the plane up to a decision speed called V1 after which he will not be able to stop the plane in case of a major incident. He executes the first block "Before V1." If no abnormal condition occurs, he executes the blocks "Before Rotation" and "Before V2 +10". If an abnormal condition is satisfied during the execution of the block "Before V1", then he can decide to execute the block "Stop the plane".

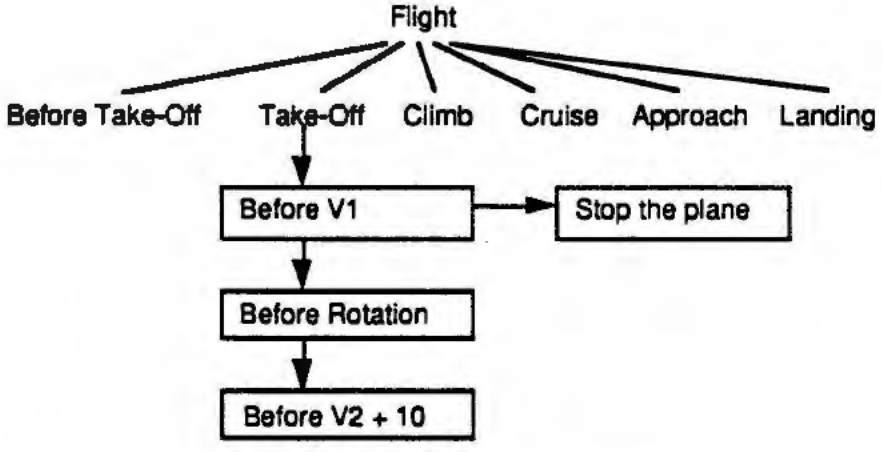


Figure 3. Simple hierarchy of contexts.

2.3 Block Execution and Context

When a context is recognized, the corresponding blocks are loaded in a short term memory (STM). The execution of a block resident in the STM is started whenever its goal is suggested and/or its triggering preconditions are totally or partially satisfied. Problem-specific criteria have to be introduced to define the term "partially satisfied". The concept of partially satisfied triggering preconditions has been introduced to handle problems occurring when there is no total match but an "almost" total match. In the case of several partially satisfied triggering preconditions, the best precondition is chosen. The execution of a block consists of performing its actions and controlling the satisfaction of the corresponding abnormal conditions. Abnormal conditions can be of two types: weak abnormal conditions which, if they occur, will cause an exit from the current block

towards another block in the same context, and strong abnormal conditions which, if they occur, will cause an exit from the current block towards another context of blocks.

For instance, a blown bulb in an aircraft cockpit may be a weak abnormal condition which does not necessitate changing the context of the flight. Conversely, an engine shut-down will cause a radical change of context, i.e., the pilot will adopt a different strategy (fly to the closest airport for example) and apply the appropriate recovery procedure.

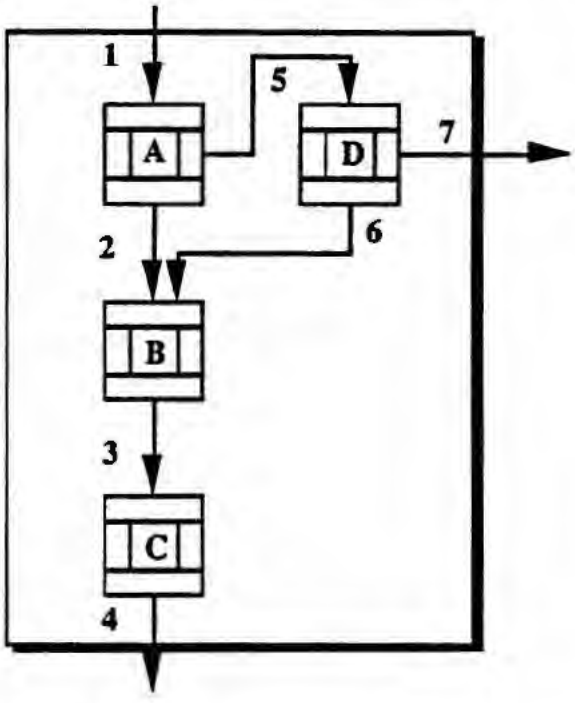


Figure 4. Context of knowledge blocks.

An example of a context of blocks is given in Figure 4 and represented in the form of a flow diagram. Let {A, B, C, D} be a set of blocks having the same contextual conditions. Arrows represent links. Links between blocks are identified by numbers, from 1 to 7 in Figure 4. For example, A and D are linked by link 5 when a specific abnormal condition of block A is satisfied. Block A is called the "root" of the context of blocks. Notice that a context of blocks is a block itself. In our example, triggering preconditions of the context block are the triggering preconditions of block A, its abnormal conditions are those of blocks D, and its goal is the goal of block C.

In the previous aviation example, each block has initially only one action that leads to the execution of a procedure. Thus, in this particular case block and procedures correspond the same documentation entities. Blocks A, B, and C stand for procedures "Before V1", "Before Rotation", and "Before V2+10". In normal situations, blocks are organized and processed in a tree sequence, e.g., A->B->C. We will call the resulting process *linear browsing* of a set of blocks. Abnormal situations interrupt this linear sequence to branch onto other blocks (generally called recovery procedures). In our aviation example, let us assume that, when applying the procedure "Before V1", the speed indicator shows an unacceptable value (weak abnormal condition 5). The pilot has to use procedure D that is to monitor the thrust except if the speed is lower than a given threshold (strong abnormal condition 7). In this case, the pilot has to change the context and "Stop the plane". If the speed is within an appropriate range, then procedure D succeeds and the pilot can apply procedure B (link 6). We will call the resulting process *non-linear browsing*, e.g., A->D->B->C.

2.4. Index Acquisition Problem

In this paper, the index acquisition problem is defined as discovery of abnormal conditions and generation of recovery actions, as well as reinforcement of current actions. It consists of two steps. It first observes the user's decisions to add abnormal conditions to a block or to reinforce current actions. If an abnormal condition has been isolated, the system either allows the user to formulate a recovery action (using a text editor-like interface), or, if possible, records the subsequent user's strategy (i.e., the trace of his/her actions) for later analysis and formulation of a recovery action from this trace. For our documentation application (described in section 3), it is possible to record the trace of user's inputs (e.g., mouse clicks or keystrokes) in the electronic documentation. In the second step, if necessary, indices are refined into a more useable form according to context. This technique is developed in section 4.

3 Application to Computer Integrated Documentation

3.1 The Technical Documentation Problem

The complexity of modern engineered systems stresses the need for "good" documentation systems. Such documentation is very labor-intensive to manage and maintain, requiring a great deal of expertise and development time. It also is needed during the actual operation of the systems, and most documentation systems (generally paper-based) are not self-explanatory.

With present documentation systems, it also is difficult to integrate different sources of knowledge in a problem solving task. This integration can be done during the problem solving process (as is done now), i.e., human operators have to construct on-line dynamic connections between various available sources of knowledge. It can be facilitated if the knowledge is prepared in an appropriate form and if the connections between the various chunks of knowledge are already programmed in the form of procedures.

Computer Integrated Documentation (CID) systems may be viewed as intelligent assistant systems which cooperate with human operators to solve operational problems. They must be able to help operators in routine and abnormal situations for retrieving information, diagnosing faults, and in some particular cases planning activities. CID systems can also be used for implementing interactive incremental knowledge acquisition techniques. As Chen and Dhar (1987) mention, to make large information banks more accessible by computer, it is best to first try and understand how reference librarians actually help users, and then to try and include these capabilities in on-line systems. The major problem in incorporating a model of user-librarian interactions into the system is the difficulty of acquiring the information for this model.

3.2 Indices as Knowledge Blocks

Using indices to retrieve a piece of information in a given document leads to the application of procedures. Consequently, an index "I" will be represented as follows: triggering preconditions (conjunction of descriptors), goals (set of possible referents), set of actions, abnormal conditions (set of conditions leading to a failure in the retrieval), and contextual conditions (set of conditions describing the environment). Each action of an index-block is represented by the set of links between triggering preconditions and the corresponding referents in a given context under normal conditions. Selecting an action is equivalent to activating a link in a network. Note that the goals of the index-block are represented by a disjunction of referents. Whenever an index I is used to retrieve a referent in the corresponding documentation, its use leads to a success or a failure.

As an index is defined by a link $L_{i,j}$ between a descriptor d_i and a referent r_j. we will introduce contextual conditions in which this particular link holds. These contextual conditions characterize the environment in which the retrieval has been made successfully. For instance, let us assume that

you want to retrieve some very specific information on the air conditioning in the main cabin of the Space Station. The first thing you may try is to select the index "air conditioning" in your documentation and browse it with this index. If you can specify the context of your retrieval, e.g., "you are a designer, you are concerned by the connection of the air conditioning system, and you only have very little information about the electrical circuitry in the cabin", then you will specify a better search in the documentation. The search will not be the same if you mention that "you are an astronaut, you are in the Space Station, and you are freezing".

4. Acquiring Indices by Experimentation

IARC (Index Acquisition and Refinement according to Context) is the prototype system that implements the technique. In this section we describe the details of the system. The current version of the PARC system is implemented on a Macintosh II Cx in Allegro Common LISP, HyperTalk and the C programming language. For this kind of knowledge acquisition from the user, an intelligent interface has been designed and developed. The program domain of expertise is the Space Station Freedom documentation needed to specify, design (and redesign), and orient the evolution of the NASA Space Station Freedom project. It currently possesses approximately 160 indices partially linked to the documentation.

4.1 Discovering Abnormal Conditions and Recovery Index-Blocks

For instance, following the same example as above, with the descriptor Descriptor-1 (triggering precondition) "air conditioning" and the contextual conditions (C_1, C_2, C_3) "you are a designer, you are concerned by the connection of the air conditioning system, and you only have very little information about the electrical circuitry in the cabin", you have four different referents in the documentation (r_1, r_2, r_3, r_4) "a list of the vendors of air conditioning systems", "a description of the air conditioning system", "a checklist of what to do when the air conditioning fails", "a diagram of the electrical circuitry in the main cabin of the Space Station". The first one is not satisfactory for you, it is a failure. The second and the third are also failures. Fortunately, the fourth one will give you the information you needed, it is a success. If this result happens frequently, you want to generate r_4 automatically from Descriptor-1 and (C_1, C_2, C_3). Unfortunately, using this block of knowledge, you observe a particular situation in which r_4 is a failure. It would be unintelligent to repeat this experience again and again, which is why you will try to add an abnormal condition to the given block. In other words, what caused your goal not to be achieved? The resulting explanation will have to be added in the form of an abnormal condition and a new block as a recovery procedure.

Inputs to the program include user judgements on the success of actual retrievals. After a referent has been found, the user can select three buttons: "success", "failure", "don't care". The system automatically records this selection by adding +1, -1, or 0 to a reinforcement slot attached to the original index-block referent inferred by the descriptor used. An example of an index-block is presented in Figure 5.

```
(TRIGG. COND.    (Descriptor-1)
(CONTEXT         (C1 C2 C3))
(REFERENTS       (G1 A1 +5 ((AC1  1) (AC2  3)))
                 (G2 A2 +2 ())
                 (G3 A3 +3 ((AC2  1) (AC3  1)))
                 (G4 A4 +2 ())))
```

Figure 5. Example of index-block.

Descriptor-1 is valid if the context conditions (C_1, C_2, C_3) are satisfied. The referents are explained generically on the first one. The first referent uses the action A_1 to reach the goal G_1; it has a reinforcement slot of +5; it has two abnormal conditions AC_1 with 1 occurrence and AC_2 with 3 occurrences. A goal is *suggested* if the user selects the corresponding action, *reached* if the corresponding action leads to a success, *not reached* if the corresponding action leads to a failure, otherwise it is *not suggested.*

The program's output is the set of index-blocks which are used as follows. When the user selects an index (i.e. a descriptor), at any point in the documentation where this descriptor is, a menu of actions (generated by a block) pops up with an ordered set of possible referents found successfully in the same context in past retrievals. The order of referents is based on the reinforcement slot. Such menus can be very different among users and other contextual conditions.

```
INPUTS:   B, status (given by the user), and inputs from the user in case of failure
OUTPUTS:  B, and a new block called newB in case of failure
METHOD:
          IF status is  success
          THEN
            reinforce(currentAction of B, +1)
                                -- if the current context is bigger than the context of B
            IF currentContext ∪ Context of B - currentContext ∩ Context of B ≠ ∅
            THEN  union(currentContext, Context of B)
          ELSE
            IF status is failure
            THEN
              reinforce(currentAction of B, -1)
              elicitAbnormCond()                          --<-- user does this
              put processAbnormCond() into currentAC of B
              incrementOccurrence(currentAC of B, +1)
            END IF
          END IF
```

Figure 6. Portion of the algorithm for the abnormal conditions acquisition

The following is a learning technique that augments index-blocks (Figure 6). The mechanism involves rote learning for incrementally capturing information about retrieval and adding it to the block contexts. The user gives information on the achievability of the goal of the procedure currently in use: e.g., *success* or *failure*. The application of a procedure is said to be a "success" when using a given block B, the result of the action is acceptable to the user. We will then say that the goal is reached. This is done in a "given" current context c. If the current context c is included in the initial contextual conditions C(B) of the block B, then a success may lead to a reinforcement of C(B). This is always the case if the user does not enter any description of the current context. However, if the user describes c, the following rules have to be considered. If the current context c and C(B) have non-empty intersection, then success may lead to either a reinforcement or an extension of C(B). If the current context c is not included in C(B), then a success leads to an extension of C(B).

If the application of the procedure is a *failure*, then an abnormal condition has to be inserted into the original set of abnormal conditions of the block. This abnormal condition is given by the user after the failure has been identified. Abnormal conditions can be seen as exceptions to the "normal" use of the block. In this case, the user returns to a manual operation and tries to provide a solution to recover from the failure. The trace of such a recovery is recorded and used for implementing a recovery block. A new block of knowledge is created and implicitly attached to the original one through an abnormal condition.

4.2 Refining Index-Blocks by Context

Using the above learning procedure huge block knowledge bases can be generated. They have to be compiled to maintain reasonable memory size and performance. Contextual conditions are used to suggest a given context of blocks. At the beginning of the knowledge acquisition process, contextual conditions are very small, i.e., they include a few very simple conditions. As the process continues, parts of some blocks can be transferred into contextual conditions (also called situation patterns) (Boy, 1987; Drummond, 1989). More generally, parts of some blocks can be transferred into other blocks upstream in the hierarchy. For instance, if a recovery block solving an abnormal condition is used often, then it cannot be qualified as a recovery block anymore and should be integrated (compiled) into the set of actions of the block(s) including this abnormal condition.

The procedure refinement mechanism has been implemented as follows (Figure 7). Assume we are applying a block B described by the contextual conditions C(B), the triggering preconditions $PC(B) \equiv PC_1(B) \& PC_2(B) \& ...$, the set of actions $A(B) \equiv \{A_1(B), A_2(B), ...\}$ associated to a set of goals $G(B) \equiv \{G_1(B), G_2(B), ...\}$, and the abnormal conditions $AC(B) \equiv \{AC_1(B), AC_2(B), ...\}$. In the CID application, this block takes the form presented in Figure 5.

```
INPUTS:    B, recoveryB, currentGoal, currentAction, currentAC
OUTPUT:    newB
METHOD:
           IF Occurrence(currentAbnormCond of B) > Threshold
           THEN
             createNewBlock
             put the name of it into newB
             putBlockInto(B,newB)
             addBlockInto(recoveryB,newB)
             removeFrom(currentGoal of B, newB)
             removeFrom(currentAction of B, newB)
             removeFrom(currentAC of B, newB)
             return newB
           END IF
```

Figure 7. Portion of the algorithm for the block refinement in context

The goal $G_i(B)$ has been suggested and consequently the action $A_i(B)$ chosen. The abnormal condition $AC_j(B)$ has been satisfied and the attached (recovery) block B' applied. If the abnormal condition $AC_j(B)$ of B is satisfied very often (i.e., its number of occurrences is greater than a given threshold) and, consequently, B' is used very often, B' cannot be qualified as a recovery procedure anymore and should be integrated into B. We will call the resulting block B^+. Two configurations are possible. First, if B' has the same contextual conditions as B, then the triggering preconditions of B' are the same as $AC_j(B)$. B^+ is then described as follows:

$$PC(B^+) \equiv PC(B) \& PC(B') \quad (1)$$

$$G(B^+) \equiv G(B) - G_i(B) + G(B') \quad (2)$$

$$A(B^+) \equiv A(B) - A_i(B) + A(B') \quad (3)$$

$$AC(B^+) \equiv AC(B) - AC_j(B) + AC(B') \quad (4)$$

and a relation between G(B') and A(B'). "-" ("+") is an operator that removes (adds) an element in a set. Second, if B' does not have the same contextual conditions as B, then $ACj(B)$ must be a conjunction of the triggering conditions and the contextual conditions of B' minus the intersection of the sets of contextual conditions of both B and B'. New contextual conditions are created as the union of the contextual conditions of both B and B': $C(B^+) \equiv C(B) \cup C(B')$. The other formulas for B^+ are the same as (1), (2), (3), and (4).

5 Example of Application of CID

It takes years of training to be a flight controller in Space Mission Control. As one part of this training, people learn to use a large corpus of documentation to solve problems. They develop a deep knowledge of the organization of these manuals in order to access the proper sections as quickly as possible. Currently the operational documentation used by flight controllers is paper-based. For the short term, the goal of CID is to help people more efficiently access documentation on a computer. One thing CID attempts to do is to help narrow the search through documents while allowing the full browsing freedom to which people are accustomed or do not have enough aid from the indexing mechanism. IARC will adapt the help CID gives according to each user's situation. CID will use IARC to incrementally learn new strategies of searching through documentation from observing people use documentation to solve problems.

As one goal is to make CID look as close as possible to documentation environment to which users are accustomed, the current organization of manuals in CID is unchanged and the pages are physically displayed the same way as in the manuals. As in regular books, they are allowed to write annotations sticked to particular words or phrases they select. They can go through the manual sequentially or by using special facilities that computers, particularly hypertext systems, allow. One of the facilities is to use descriptors to go directly to a location in the manual where that descriptor is located. Another is to go directly to a particular section or page if they know where they want to go. They need not to leaf through pages to find a certain section. This alone makes the process of searching through documentation easier.

With this tool, when the user clicks on a descriptor, a menu pops up providing a list of possible locations where the desired information might be located. The user may select a menu item and CID will provide the corresponding section to him/her. If the user is satisfied with this information, then he may click on "success", this will reinforce the persistence of the selected item. If he is not satisfied, he may click on "failure", this will penalize the persistence of the selected item. Otherwise, the system will take into account this attempt. Of course, each time the use cooperates (i.e. tells the computer to reinforce or penalize the persistence of an item), the corresponding context is taken into account by IARC which update the corresponding index-block in CID. This way, IARC provides an intelligent assistant capability to CID. By acquiring more index-blocks and especially more context, CID should assist flight controllers in their search through documentation in an intelligent fashion.

In the example shown in Figure 8, the context was already known from CID and described by a set of strings such as Name_of_the_user (Guy) and Type_of_task_leading_to_the_retrieval (Browsing). The user has clicked on the descriptor "OPS RECORDER", then CID provided the following menu (that is not displayed on the figure):

11.2. S-BAND_UHF LAUNCH REQUIREMENTS
11.46. LOSS OF RECORDERS

The user selected the item "11.2. S-BAND_UHF LAUNCH REQUIREMENTS". CID displayed the corresponding page. The user did not like the corresponding information and he said that, i.e. he clicked on "failure" (in other words, the goal was not reached for the executed index-block). Then CID provided the following menu:

11.46. LOSS OF RECORDERS
I AM LOST !!!

because the user may have wrongly selected the item "11.2. S-BAND_UHF LAUNCH REQUIREMENTS" (the system gives another chance to the user). If the user selects another location, the same process will take place. If the user selects "I AM LOST !!!", CID provides the opportunity to return to either the table of contents or the table of indices, or to change the current description of the context.

6 Relation to Other Work

Index knowledge acquisition is related to procedural knowledge acquisition because of their common interest in the way actions are handled by users according to context. Acquisition of procedural knowledge from domain expert has been explored earlier in the MOLGEN system (Friedland, 1981).

A considerable effort has been already invested in modeling procedures. PRS (Procedural Reasoning System) work (Georgeff & Lansky, 1986) is very similar to ours. However, the notion of context in the knowledge areas (KA) of PRS is slightly different than the one used in the block KR. PRS does not currently have the notion of abnormal condition. Other work in this direction includes architectures which try to couple sensing to acting. Nilsson described new techniques for organizing actions in dynamic environments in which the effect of actions and the state of the environment can not always be reliably predicted. His solution to the problem is based on hierarchical "action networks". In this representation, the entire network and each of its hierarchical components act like homeostatic servo systems attempting to maintain the environment in specific states. From this perspective, Nilsson's action nets (Nilsson et al., 1990) and contexts of blocks are almost identical in nature. We introduce here object and link concepts to improve naturalness, implementation, and connectivity of blocks. Drummond has shown how to compile reactions from specifications (Drummond, 1989).

From the perspective of differentiating contextual conditions from triggering preconditions, similar ideas have been already developed in AGE (Nii & Aiello, 1979). In BB1 and GARDIAN, a similar representation has been implemented (Hayes-Roth, Washington, Hewett & Hewett, 1989).

From the perspective of representing insufficient or incomplete procedures, both humans and computers must be able to react promptly to new information, and they must be able to change or repair their knowledge when new information produces contradictions or when initial assumptions are withdrawn. Explicit representation of abnormal conditions or exceptions provides a simple mechanism for capturing knowledge when we discover it. Variable precision logic is the closest representational mechanism to the blocks (Michalski and Winston, 1986). Similar approaches have already been described in (Winston, 1983) and (Williamson, 1986).

From the perspective of knowledge refinement, the basic idea of building macro-operators (set of actions in this paper) was well-defined a long time ago in STRIPS (Fikes, 1971). The block representation differs from STRIPS by the fact that we introduce the concepts of contextual conditions and abnormal conditions which improves efficiency and facilitates knowledge acquisition. Further work has been done on macro-operators such as Korf's work exploiting problem decomposability and providing heuristic utility metrics for deciding whether a new macro-operator is likely to increase the overall problem-solver performance (Korf, 1985). Recently, a contribution has been provided by Shell and Carbonell towards a general framework for composing disjunctive and iterative macro-operators (Shell & Carbonell, 1989).

7 Discussion And Future Directions

The current practice of design engineering is often machine-centered: optimized for the tasks that can be performed by the machine itself, and neglecting to support the complementary role of the

human in the loop (Boy & Gruber, 1990). The approach described in this paper takes an alternative view in which the design of indices to retrieve information is based not only on design rationale, but also on users requirements and suggestions when they are operating systems. In this view, the user is a functional component of an intelligent system.

Formalization of the contextual conditions is still problematic. On the one hand, the contextual conditions should be minimal to avoid excessive calculations. On the other hand, they must include as much information as possible to characterize the current situation. Definition of the contextual conditions is problem dependent. It is very difficult, if not impossible, to define such contextual conditions *a priori*., since context comes from experience using the designed system. However, contextual conditions can be incrementally augmented, i.e., expressions may be added to the current contextual conditions each time a new success is found. An expression can be either a fact or a condition from the external environment.

Currently, we are trying to answer several questions about our technique. First, is there a "good" threshold for deciding to create a new block, delete an "unused" one, or include an action in a block? Second, what is the "good" level of detail to model the contextual conditions and the abnormal conditions for dissociating referents from the same descriptor? We are now studying many different user requirements and suggestions, searching for the minimal number of slots for these entities.

Sometimes, the design of abnormal conditions is not straightforward. The user has to search (and backtrack) tediously in the documentation before finding a recovery index. In our technique, an intelligent (or, by default, an exhaustive) search has to be performed, in order to retrieve a referent which would be accepted by the user. We are currently working on the implementation of such a search mechanism as a general recovery procedure.

Current results have shown that hypertext is a good programming tool for the development of such systems. However, although hypertext systems increase accessibility, they do not provide any built-in selectivity mechanism. Hypertext systems are also known to have problems: (1) *disorientation*, i.e., the tendency to lose one's sense of location and direction in a nonlinear document; and (2) *cognitive overhead*, i.e., the additional effort and concentration necessary to maintain several tasks or trails at one time (Conklin, 1987). In other words, while non-linear or hypertext systems may dramatically increase the accessibility of information, this increased accessibility may magnify an already severe problem of selection (Jones, 1987). For these reasons, our knowledge-based systems technology can be very helpful for alleviating the selection problem and cognitive overhead of the user. Another reason for developing the block representation on top of hypertext systems is an easy mapping between the two: i.e., blocks are naturally associated with hypertext nodes. This is the main goal of our research effort on context-sensitive retrieval procedure acquisition.

The IARC system requires additional research to become fully operational. Future work will include the development of a learning mechanism generalizing the cases acquired during the use of the system.

Acknowledgements

Thanks to Philippa Gander for many useful comments on this paper. Nathalie Mathé and I developed premises of these ideas together when I worked in the AI Research Group of ONERA/CERT. Thierry Caminel and Frédéric Vaute deserve special acknowledgements for an early implementation of these concepts on telerobotics application at Dialogics in Toulouse. I am very grateful to Catherine Baudin, Tom Gruber and David Thompson who gave me some very pertinent comments on the block knowledge representation. Mark Drummond, Peter Friedland, Jody Gevins, Smadar Kedar, Amy Lansky, and Monte Zweben also provided astute advice towards improving the quality of this paper.

References

Anderson, J.R. (1983). Knowledge Compilation: The General Learning Mechanism. *Proceedings of the International Machine Learning Workshop*, Monticello, Illinois,June, pp.203-212.

Boy, G.A. (1987). Operator Assistant Systems. *Int. J. Man-Machine Studies*, 27, pp. 541-554.

Boy, G.A., & Mathé, N. (1989). Operator Assistant Systems: An Experimental Approach Using A Telerobotics Application. *IJCAI Workshop on Integrated Human-Machine Intelligence in Aerospace Systems*, Detroit, Michigan, USA, August 21.

Boy, G.A., & Gruber, T. (1990). Intelligent Assistant Systems: Support for Integrated Human-Machine Systems. AAAI Spring Symposium on Knowledge-Based Human Computer Communication, Stanford, March 27-29.

Chen, H., & Dhar, V. (1987). Reducing Indeterminism in Consultation: A Cognitive Model of User/Librarian Interactions. Proceedings of AAAI'87, Seattle, WA.

Conklin, J. (1987). Hypertext: An Introduction and Survey. Computer, September.

Drummond, M. (1989). Situated Control Rules. *Proceedings of the First International Conference on Principle of Knowledge Representation and Reasoning*, Morgan Kaufmann Pub., Toronto, May.

Fikes, P.E., & Nilsson, N.J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2, pp. 189-208.

Friedland, P. (1981). Acquisition of Procedural Knowledge from Domain Experts. Proceedings of IJCAI'81, Kaufmann, Los Altos, CA, pp. 856-861.

Georgeff, M.P., & Lansky, A.L. (1986). Procedural Knowledge. *Proceedings of the IEEE Special Issue on Knowledge Representation*, 74, pp. 1383-1398.

Hayes-Roth, B., Washington, R., Hewett, R, & Hewett, M. (1989). Intelligent Monitoring and Control. *Proceedings of IJCAI*, Detroit, Michigan.

Jones, W.P. (1987). How do We Distinguish the Hyper from the Hype in Non-Linear Text ?. *Human-Computer Interaction - INTERACT'87*, H.J. Bullinger and B. Shackel (Editors), Elsevier Science Publishers, North Holland.

Michalski, R.S., & Winston, P.H. (1986). Variable Precision Logic. *Artificial Intelligence*, 29, Elsevier Science Publisher, pp. 121-146.

Nii, H.P., & Aiello, N. (1979). AGE: A Knowledge-Based Program for Building Knowledge-Based Programs. *Proceedings of the 6th International Joint Conference in Artificial Intelligence*, Kaufmann, Los Altos, CA, pp. 645-655.

Nilsson, N.J., More, R., Torrance, M. (1990). ACTNET: An Action-Network Language and its Interpreter. (A Preliminary Report), AAAI-90 paper draft, Computer Science Department, Stanford University, Stanford, CA 94305.

Rosenbloom, P.S. (1983). *The Chunking of Goal Hierarchies: A Model of Practice and Stimulus-Response Compatibility*. Doctoral Dissertation, Carnegie Mellon University.

Shell, P., & Carbonell, J.G. (1989). Towards a General framework for Composing Disjunctive and Iterative Macro-Operators. *Proceeding of IJCAI*, Detroit, Michigan.

Williamson, K.E. (1986). Learning from Exceptions in Databases. in *Machine Learning: A Guide to Current Research*, edited by Tom M. Mitchell, Jaime G. Carbonell, and Ryszard S. Michalski, Kluwer Academic Publishers.

Winston, P.H. (1983). Learning by Augmenting Rules and Accumulating Censors. *Proceedings of the International Machine Learning Workshop*, Monticello, Illinois,June, pp. 22-24.

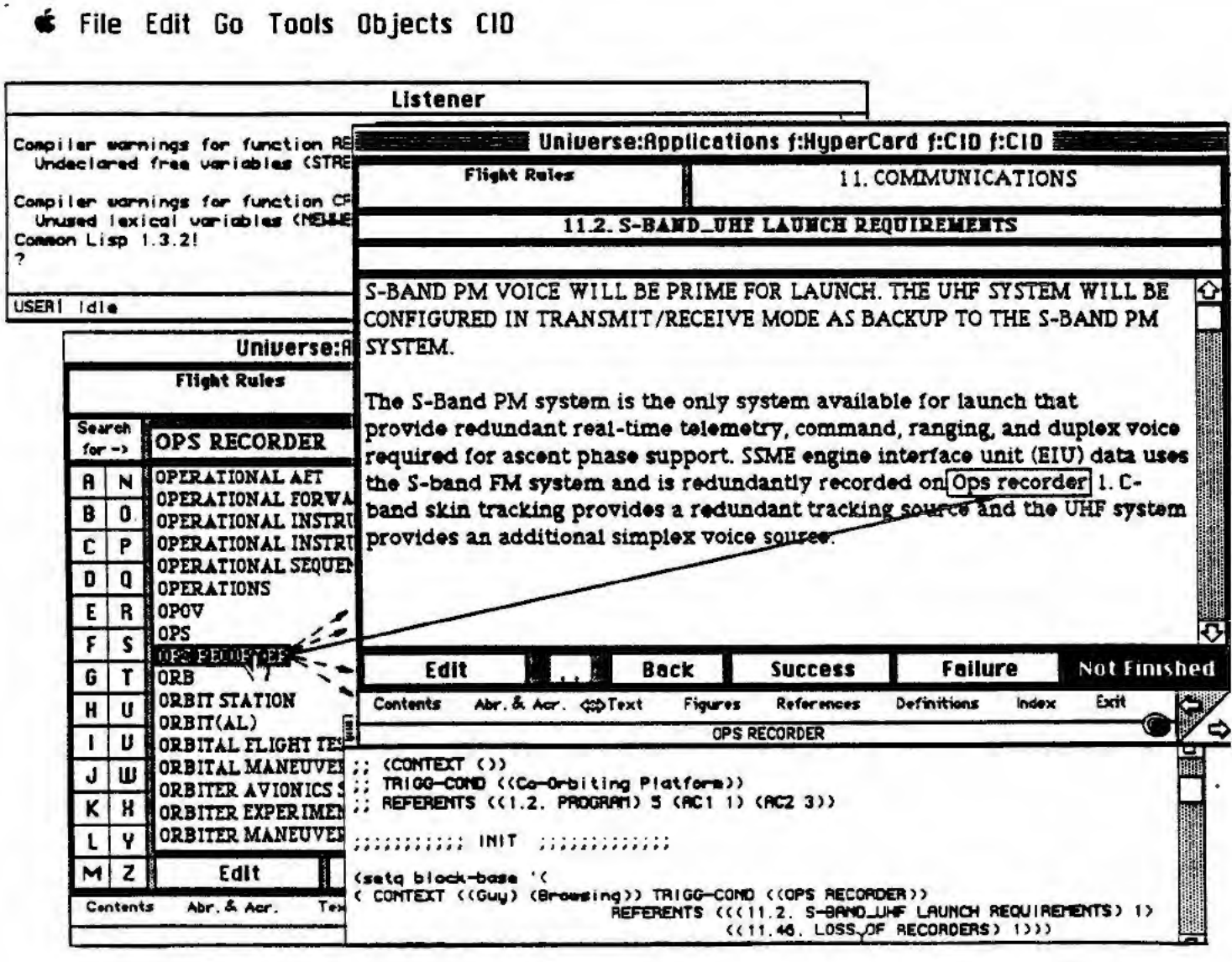


Figure 8. Example of screen of the current CID system.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

MEASURING THE STRUCTURE OF DIAGNOSTIC EXPERTISE USING PATHFINDER

JOHN H. BRADLEY

Department of Decision Sciences
East Carolina University, Greenville, NC 27858

LAWRENCE L. SCHKADE

Information Systems Department
University of Texas at Arlington, Arlington, TX 76019

The Pathfinder technique of eliciting cognitive structures holds great promise as a knowledge acquisition tool since it provides unbiased, numerical and graphical representations of the detailed concepts and relationships that are being used by the expert to solve a problem. The original study that was used to refine the Pathfinder technique was replicated in this study using machine diagnosticians. A revision of one of the basic assumptions in the original study led to improvements in the implementation of the technique in which an independent measure of cognitive skills was used to verify the presence of expertise in the subjects. More research is needed to refine the implementation methodology of this powerful technique.

INTRODUCTION

An expert is someone who is characterized by superior performance within a specific domain of activity (McGraw & Harbison-Briggs, 1989, Johnson, 1987, Van Gigch, 1978). That superior performance is dependent not only on the domain experience and training of the individual, but it also depends upon intimate familiarity with how the domain entities can relate to one another in a problem situation (Brehmer, 1980). It is knowledge of these relationships that allows the expert to effectively reason about a problem; to traverse the mental distance from symptom to solution. Experts therefore have the mental ability to organize the entities and relationships into a structure that permits effective and accurate reasoning (Keravnou & Johnson, 1986). They are able to construct a mental network, or map, of these relationships and facts that provides a path for their reasoning from initial problem formulation to a quality solution (Johnson, 1987). If the expert system is to emulate expert reasoning, then this expert knowledge structure must be accurately elicited and represented in the knowledge base.

One of the major problems involved in the elicitation of expert knowledge structure is expert-induced bias. Bias in expert knowledge can be due to several factors. An expert tends to teach a novice the way the expert wants the novice to perform the task based on the novice's limited capabilities, not the way the expert himself performs it. Since the knowledge engineer or knowledge elicitation tool can easily be viewed as a novice, the expert frequently shifts into a "teaching" mode and introduces bias into the knowledge. This kind of bias is also introduced when the expert is discussing a task that involves tacit knowledge. If the expert is not sure just how he performs the task, then he must rely on knowledge of how it should be performed. Bias is also introduced when the domain knowledge is highly cognitive and/or very complex.

In this situation, the English language may not be sufficient to express thought processes or involved procedures. In translating mental processes into the English language, not only is some knowledge lost, but there is also a bias toward explanations that are more easily translatable; explanations that the knowledge engineer is more likely to understand. In all of these situations, relationships are adjusted to meet the communication requirements, which means that the structure of the knowledge is being modified in the elicitation process. Using indirect methods (Olsen & Rueter, 1987) to elicit knowledge structure is one solution to this problem.

Many of these indirect techniques involve the elicitation of independent judgements concerning pairs of concepts. A concept can be anything from a quality that an entity may posses to a component of a machine. Cooke and McDonald (1988) discuss several psychological scaling techniques that produce structural representations of knowledge from sets of human judgements. They contrast Multidimensional Scaling, a technique for identifying global information, to Pathfinder, a technique for capturing local relationships with in the data.

The Pathfinder technique was developed by Roger Schvaneveldt et al. (1985) as a means of constructing a detailed representation of an expert's mental structure. This technique involves the conversion of a set of relatedness judgements into psychological distances, resulting in a fully connected, weighted network of concepts and relationships. The Pathfinder algorithm is applied to this network to generate a reduced network (PFNET) containing only the shortest paths. The PFNETs can be represented both numerically and graphically. If the expert traverses his mental map by using the strongest relationships first (concept pairs with the least psychological distance between them), then the PFNET should provide a wealth of detailed structural information for the knowledge engineer. The original study, as reported by Schvaneveldt et al. in the International Journal of Man Machine Studies, was replicated to see if the Pathfinder technique is valid in other domains and to test improvements to the technique. Improvements were required due to apparent errors in the fundamental assumptions of the original design. Reporting the results of this replication and the necessary improvements to the Pathfinder technique is the task of this paper.

THE PATHFINDER MODEL

Link-weighted networks have been used successfully in psychology and artificial intelligence to represent conceptual models. Recently, concept pair scaling methods that yield general network structures have been developed, but the resulting networks are often difficult to analyze. They generally do not support detailed numerical analyses. The Pathfinder algorithm has been used to reduce these general networks into concise structures that have been shown to be accurate representations of the mental structures of a human problem-solver (Cooke and Schvaneveldt, 1988; Schvaneveldt et al., 1985). A Pathfinder network (PFNET) is a link-weighted network that is derived from estimates of dissimilarity (distances) for pairs of entities or concepts. The essential idea is that "dissimilarities between entities should be represented as arcs in a PFNET if the resulting arcs form the minimum weight paths" (Schvaneveldt et al., 1985). Estimates of dissimilarity are obtained by converting the concept pair judgements of similarity, or relatedness, to distance measures of dissimilarity. Given a similarity judgement, S_i, and a maximum judgement rating, MaxS, the dissimilarity distance, D_i, is derived by

$$D_i = \text{MaxS} - S_i$$

Using a foundation in graph theory, PFNETs are derived by regarding the matrix of dissimilarities as a network adjacency matrix, computing the r-distance matrix, using the Minkowski r-metric of the adjacency matrix in order to accommodate the different assumptions about the nature of the measurement scale (eg. ordinal, interval, ratio), and reducing the matrix by eliminating each arc that has weight greater than the r-distance between the nodes connected by the arc. PFNETs have been shown to have significant predictive power in categorizing

expert and novice USAF fighter pilots, software programmers, and others (Cooke and Schvaneveldt, 1988; Schvaneveldt et al., 1987; Schvaneveldt et al., 1985).

One problem with networks derived from psychological measures of concept pair distances is the assumption that these distances meet the condition for triangle inequality. Psychological judgments may show systematic violations of the triangle inequality (Tversky, 1983). For one thing, entities may be related to one another in different ways. Tversky uses the example where Jamaica is similar to Cuba and Cuba is similar to Russia, but Jamaica is not at all similar to Russia. One way to avoid this condition is to regard dissimilarities as arc weights in a network (see Schvaneveldt, et al., 1988, for more detail).

In this study, the arc paths that are used by an individual to traverse the mental network of concepts is of primary interest. Since one characteristic of expertise is cognitive speed (Johnson, 1983), the shortest path between related concepts is this path. In graph theory, the minimum distance between two nodes is the minimum weight of all possible paths in which the weight of a path is the sum of the weights of the arcs in the path. In this case, it is necessary to eliminate some of the arcs in order to identify those paths that will most likely be used by the expert. This will require some other distance measure that will allow different assumptions about the level of measurement associated with the dissimilarities.

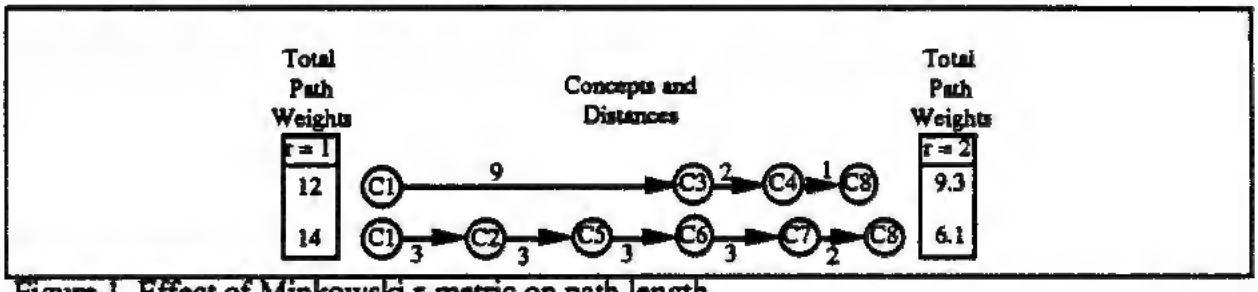


Figure 1 Effect of Minkowski r-metric on path length

For example, given two paths of concepts (C*n*), (C1-C3-C4-C8) and (C1-C2-C5-C6-C7-C8), and MaxS=9, summing the path weights given in Figure 1 results in the first path having the shortest length (9+2+1=12). However, this path will never be taken since C1 and C3 are totally unrelated. A measure is required that will compensate for large psychological distances between concepts.

One distance measure that fits this requirement is the Minkowski distance measure which can be adapted to measure paths within networks. Given a path consisting of arc weights, W_i, (W_1+W_2) becomes $(W_1^r+W_2^r)^{1/r}$, $r > 0$. The path length is the sum of these distances. Increasing the value of r increases the relative contribution of the larger weights in a path. In Figure 1, raising the value of r to r=2 changes the path lengths so that the second path now has a shorter path length.

Schvaneveldt et al. (1988) provide a psychological interpretation of the requirement for the Minkowski r-metric.

> With r=1, all components (arcs in a path) have equal weight in determining the weight of a path. As r increases, the components with greater magnitude receive greater weight until, in the limit, only the largest component (arc) determines the weight of a path. One psychological interpretation of larger values of r is that the perceived dissimilarity between entities is determined by the dissimilarity of the most dissimilar relations connecting the entities. (p. 341)

The r-metric was used to develop the Pathfinder algorithm to remove unused arcs from the network.

The Pathfinder networks are the central representations that will be analyzed in this study. The methodology that follows is primarily concerned with replication of the original study. The resulting PFNETs are analyzed for qualities that one would expect to find in the mental model of an expert machine diagnostician.

THE ORIGINAL STUDY

The original study was conducted at a USAF fighter pilot training school. Subjects were classified into two groups; instructors and students. The seven instructors were considered to be the experts because they had more flying experience than the students. The instructors averaged 2,583 flying hours while the seventeen students only averaged 200 hours of flying time.

The researchers defined expertise in terms of superior performance but did not consider performance as a criteria for expert identification except under the assumption that superior performance is required in order to maintain a position as an instructor. They assumed a strong correlation between performance and years of experience. This assumption may not be valid. It is conceivable that an instructor could be retained in that position based on only average flying performance but excellent teaching performance. Should such a person be considered an expert pilot without some means of validation of his flying performance? Evidence of this very situation was found in the replication of this study which invalidated any subsequent conclusion.

In the original study, the instructor PFNET contained fewer total links (relationships) within a set of thirty concepts (nodes) than were found in the student PFNET (refer to the Schvaneveldt et al. paper for these representations). Since the instructors were considered to be the experts, the researchers interpreted this to mean that experts have more highly organized mental structures than novices (the students) and thus retain fewer bits of more highly refined knowledge in their mental structures.

THE CURRENT STUDY

The subjects that were used for this study were instructors and students at the United States Postal Service (USPS) Technical Training Center in Norman, Oklahoma. The Training Center trains technicians to maintain the mail handling machines used by the postal service. Diagnostic skills are essential for these maintenance personnel since down time for the complex and expensive mail handling machines relates directly to the cost of handling the mail.

The instructors and students were all involved in the maintenance of a mail sorting machine that is referred to as the OCR. Maintenance technicians for the USPS promote through ten technical levels. The technicians selected for this study are all level 8 through 10 and are referred to as Electronic Technicians (ET). Technicians are assigned to a site where the machines are being used just prior to attending the OCR Maintenance course at the Technical Training Center. Some of these technicians have experience maintaining other electromechanical equipment while others have only brief electronic experience and almost no formal or vocational education.

Due to the fact that promotions are based mostly on seniority and other organizational quirks, the sample group contained a good mix of novice and expert diagnosticians. On the other hand, these people have self-selected this profession and have enough diagnostic ability to maintain their jobs. This means that the differences between the experts and the novices were not as great as they would have been between the experts and a group of naive maintenance workers. The subjects were classified into expert and novice groups based on their position as either instructor or student. The instructors had several years of experience with the OCR while the students had only a month or two, if any.

Using input from a panel of top rated technicians, a set of seventeen concepts was carefully chosen. A computer was used to randomize all possible pairs (136 pairs) of these concepts. The subjects were given a questionnaire containing two major sections. The first section elicited their years of experience in troubleshooting electro-mechanical equipment and level of education. The second section presented the list of concept pairs with a ten point Likert scale for each pair which ranged from Highly Related to Unrelated. Instructions were derived from the original study and required the technician to indicate how closely related the two concepts are in the context of a specific problem scenario.

Seven instructor responses and fifty-nine student responses were useable. These relatedness scores were converted into psychological distance measures and averaged for the instructor and student groups resulting in two vectors of 136 distances, each vector representing the average subject in each group. Each vector was then converted into a matrix for input into the Pathfinder program. Schvaneveldt provides a software package that converts a matrix of distances into a Pathfinder network. The Macintosh version generates graphical representations of these networks.

The Pathfinder algorithm was applied to each matrix to generate a Pathfinder network representation for both the instructor (Figure 2) and the student (Figure 3) technicians. A greater number of connecting links is an indication that the concept is central to the paths through the network. In other words, those concepts are more important in the cognitive processes in the group than the concepts that only have one or two links. In the PFNETs, the letter following the concept number designates whether the concept is Abstract or Concrete. The PFNETs have been subjectively arranged starting with the concepts that have the greater number of links at the top and then arranging the remainder of the network in as close to a hierarchical structure as possible. Notice that the student PFNET emphasises both abstract and concrete concepts. This indicates the heterogeneity of the student group and is addressed latter in this study. The instructor group contained only seven instructors and did not reveal as much as the fifty-nine technicians in the student group.

In this study, seven instructors and fifty-nine students completed the questionnaire with 136 concept-pair distances. An ANOVA analysis on the resulting PFNETs showed a significant mean difference ($p<.0001$; $F=50.87$) in the average instructor and student responses. This suggests that the two sets of responses are from different populations. The instructor PFNET contained twenty-four links while the student PFNET contained thirty-three links. This is the same relationship that was observed in the aircraft pilot PFNETs in the original study.

In order to perform an accurate comparison of the two groups, the PFNETs must be paired by concept and the number of links that are attached to each concept compared. One link would then count as two attachments. The Wilcoxon Signed Rank Test (Clark & Schkade, 1979) was used to detect differences in the frequency of attachments to each concept. After eliminating the concepts that have an equal number of attachments, the remaining student concepts contained sixty-one connecting links which is significantly greater ($p<.0179$; $z=-2.1015$) than the forty-three connecting links remaining in the instructor PFNET. The significance of this finding may be questioned because of the small number of instructors in both studies. However the students did have more links in both studies which would be the expected result if the method that was followed in this study replicated the original study.

In contrast to the original researchers' explanation, one would expect the instructor PFNET to contain more links, not fewer. This seems reasonable since experts usually exhibit a command of more knowledge in their domain than do the novices. In order to further validate the technique as one which represents expert knowledge, several additional analyses were

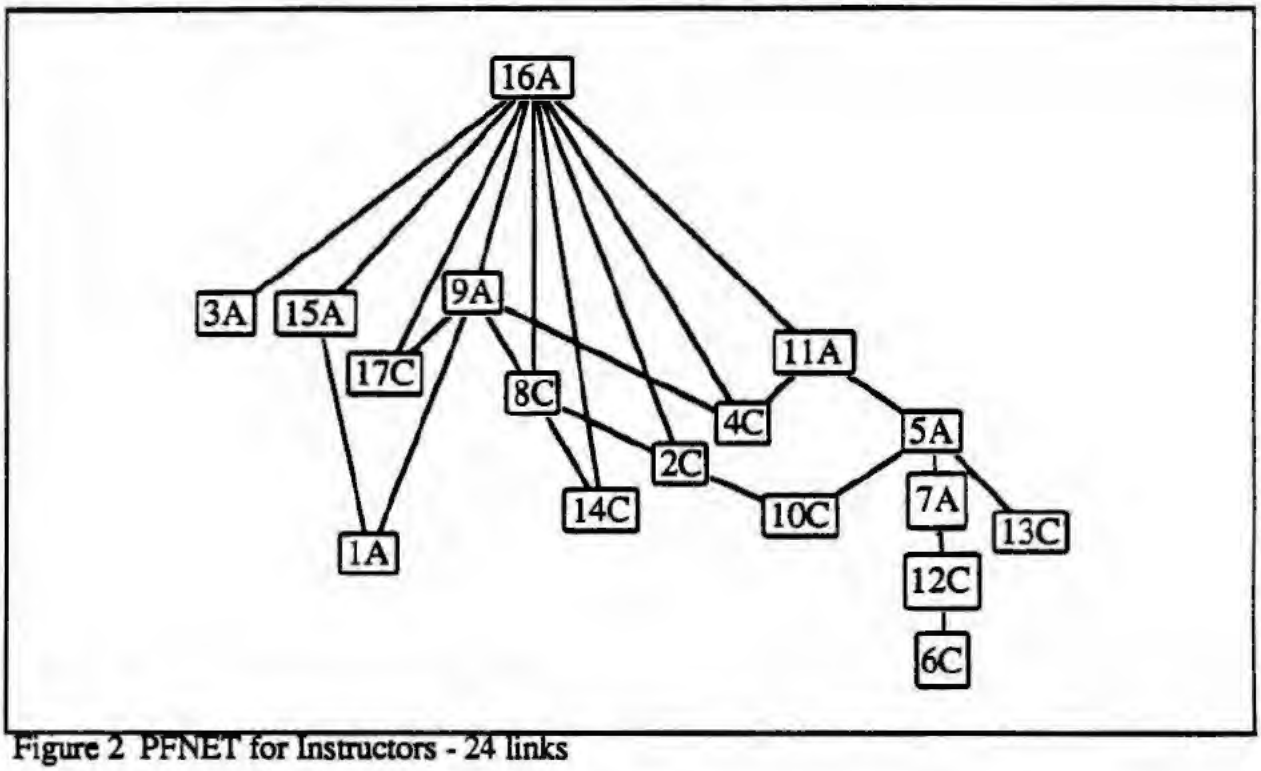


Figure 2 PFNET for Instructors - 24 links

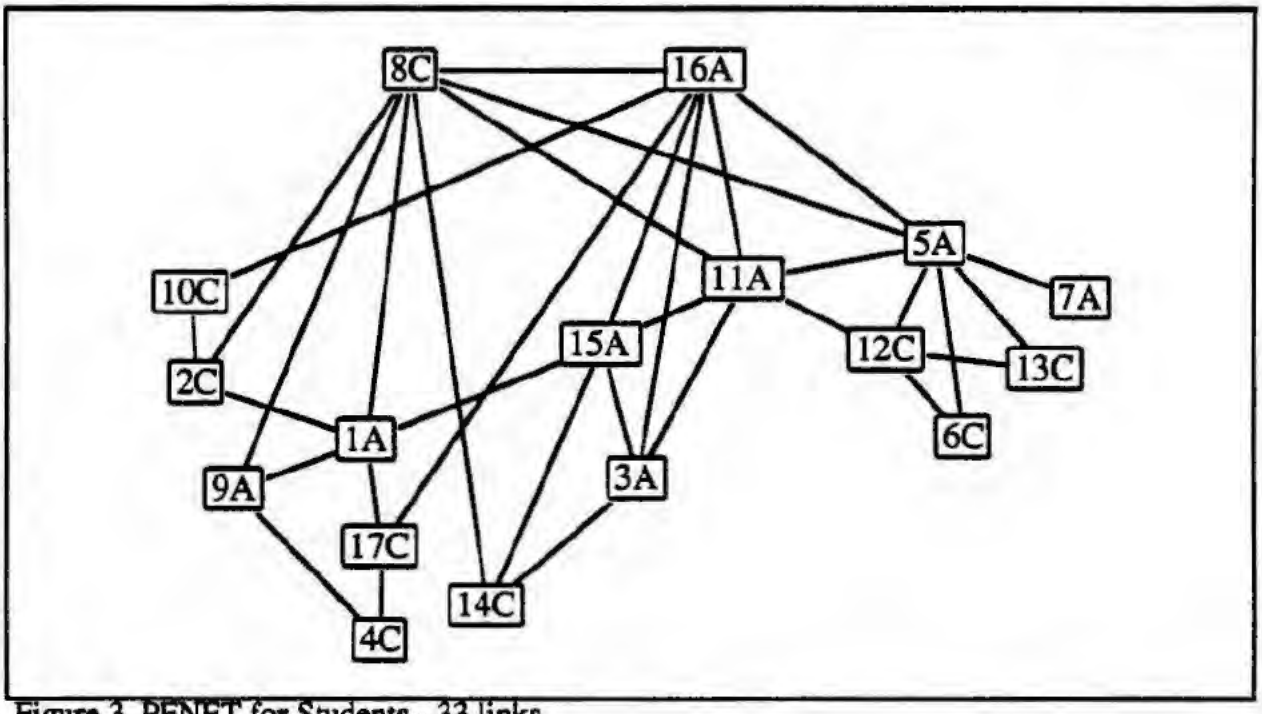


Figure 3 PFNET for Students - 33 links

conducted. Four characteristics that are common to experts were selected. These characteristics are holistic perception, critical thinking, superior performance, and the use of abstract concepts. The PFNETs were analyzed for indications of these characteristics in the instructor networks.

Holistic perception

Holistic perception refers to the way an expert views a problem. For example, a novice machine diagnostician will usually select one symptom and analyze that symptom until a cause is detected. On the other hand, an expert will consider the events prior to the observance of the symptom, the time of day, the temperature in the room, the machine functions that are affected, patterns in the symptoms, and many other variables in developing a plan to proceed with the investigation. One would expect that these differing formulation schemes would result in a larger number of links in the instructor PFNET since the expert is perceiving more relationships than the novice. The instructors had significantly fewer links in their PFNETs than the novices. As discussed above, the Wilcoxon Signed Rank Test (Clark & Schkade, 1979) was used to detect differences in the frequency of the links attached to the concepts. After eliminating the concepts that have an equal number of links, the remaining student concepts contained sixty-one connecting links which is significantly greater ($p<.0179$; $z=-2.1015$) than the forty-three connecting links remaining in the instructor PFNET.

Critical thinking

A pilot study revealed a strong correlation between scores on the Watson-Glaser Critical Thinking Appraisal, a measure of critical thinking ability, and judgements of performance in machine diagnostics. The Critical Thinking Appraisal is an independently verified instrument (Jaeger & Freijo, 1975, Simon & Ward, 1974) that was designed to measure the composite of attitudes, knowledge, and skills that results in the ability to correctly perform the tasks of problem definition, information selection, assumption recognition, hypothesis generation, and conclusion of valid inferences (Watson & Glaser, 1980). In this study, this instrument was found to be an very good indicator of diagnostic ability. The instructors were expected to score higher on the Critical Thinking Appraisal than the novices. Thirty-two Critical Thinking scores were analyzed using a dummy variable to indicate whether a subject was an instructor (experienced) or a student (inexperienced). This analysis was inconclusive ($p<.20$, $R^2=.05$, $r=.23$) revealing an insignificant correlation between experience and critical thinking ability in this sample.

Superior performance

Schvaneveldt et al. (1985) and Cooke and Schvaneveldt (1988) both referred to the characteristic of superior performance as a primary indicator of expertise. Since expertise is usually defined in terms of superior performance (Johnson, 1983, Van Gigch, 1978), one would expect the more experienced instructors to exhibit this superior performance in problem situations. The training facility contained a machine lab in which the instructors planted faults. The students were given ten trials throughout the class in which several instructors observed each student's approach to the problem and the time it took the student to resolve the malfunction. Over several years, the instructors would regularly face diagnostic situations. These occured at times when the machines would develop real malfunctions. During these times, the lead instructor would observe the diagnostic performance of the instructor who fixed the machine. Ratings of diagnostic performance were accumulated using a five-point scale. A rating of '5' for consistently outstanding performance and a rating of '1' for consistently poor performance. In this study, it was interesting to know if experience alone was an indicator of diagnostic performance. In other words, can a person have a large amount of experience and yet display only adequate performance?

A total of sixty-nine technicians responded to questions concerning their years of experience maintaining electromechanical equipment. Their experience was divided into two segments; experience with USPS equipment and experience maintaining any other equipment. These two quantities were totaled to generate a third variable; total experience. The validity of these responses can certainly be argued since these experiences may not have been relevant to the research scenario. There is no practical way to verify their relevancy.

A correlation analysis with all three experience variables resulted in only total experience being significantly related ($p<.0391$, $F=4.44$, $R^2=.066$) to performance. John and Ronda Hunter

(1984) performed a meta-analysis of the cumulative research on various predictors of job performance and found that there was no predictor better than cognitive ability. Over thousands of studies, cognitive ability had a mean Pearson's correlation coefficient of .55 across "all known job families". The coefficient decreases corresponding to the cognitive complexity of the job requirements from .61 for cognitively complex jobs to .27 for cognitively simple jobs. Since machine diagnostics is probably in the middle range of cognitively complex tasks, a correlation coefficient of .50 or greater was selected as the indicator of significant correlation. Therefore, an r of .26 is not high enough to say that experience is a significant measure of performance. Postal experience was not significantly correlated with performance ($p<.3425$, $r=.12$).

It is interesting to note that when total experience is combined with critical thinking, for the thirty-two subjects who took the Critical Thinking Appraisal, the relationship to performance increased to $R^2=.732$, $r=.86$. This suggests that superior performance in this study is a composite of cognitive ability first, then experience.

Use of abstract concepts

The last characteristic to be analyzed is the use of abstract concepts. Since experts chunk or group their knowledge differently than novices (Cooke and Schvaneveldt, 1988; Ettenson, Shanteau, and Krogstad, 1987; Glaser, 1984; Freyd, 1983; Kahney, 1983; Bouman, 1982), the expert's mental model should contain different relationship strengths than the novice's. The novice uses the inefficient approach of focusing on the physical components instead of on the functional components and their information flows (Tenny & Kurland, 1988). This has been demonstrated in prior mental model research (Cooke and Schvaneveldt, 1988; Tenny & Kurland, 1988; Ettenson, Shanteau, and Krogstad, 1987; Glaser, 1984; Freyd, 1983; Kahney, 1983; Bouman, 1982). Cooke and McDonald (1988) found that levels of abstraction can be obtained from the PFNET representations. In this study, abstract concepts such as Patterns of Symptoms(9A) and the Interpretation(16A) function should be more central to the reasoning of the experts. Conversely, concrete concepts such as Master Reader Cabinet(8C) should be more central to the reasoning of the novices. These relationships were observed in a pilot study.

The concepts that are included in most of the paths through a network are more likely to be used and are thus more important in the cognitive task of diagnosing a machine fault. The concepts that are included in only one or two paths are not as useful in the reasoning process. Path inclusion can be measured by the number of links attached to a concept. The abstract concepts, therefore, should have more connecting arcs in the instructor PFNET while the student PFNET should display more connecting links for the concrete concepts. The more connecting links, the more important the concept is in the reasoning process (Cooke & Schvaneveldt, 1988).

Eight of the seventeen concepts chosen for this study are abstract in the sense that they either pertain to reasoning about the relationship between multiple machine functions or they refer directly to those machine functions. Several of the concepts were explicitly designated as being abstract to avoid misinterpretation by the technician. In contrast, the remaining concrete concepts refer to specific machine components. Using the Wilcoxon Signed Rank Test, the instructors had significantly fewer ($p<.0465$, $z=-1.68$) abstract relations than the students. The instructors had twenty-eight abstract connecting links while the students had thirty-eight.

The results of this part of the study indicate that not all of the instructors were experts, even though they had more experience than the students. This finding is important. Using experience as the classification variable in this study led to incorrect and misleading conclusions. To avoid this problem, other expert/novice research designs should include some meaningful validation of the assumpion that the "expert" subjects actually possess expertise. Expert system development methods and tools should not be developed based on trials with individuals that may not be experts in their domain.

IMPROVEMENTS TO THE ORIGINAL STUDY

Using the performance ratings as a classification variable, the technicians in this study were regrouped. The technicians that were rated as outstanding were placed in the expert group while the technicians that were rated as being consistently poor in their diagnostic performance were placed in the novice group. Technicians that were rated between these extreme ratings

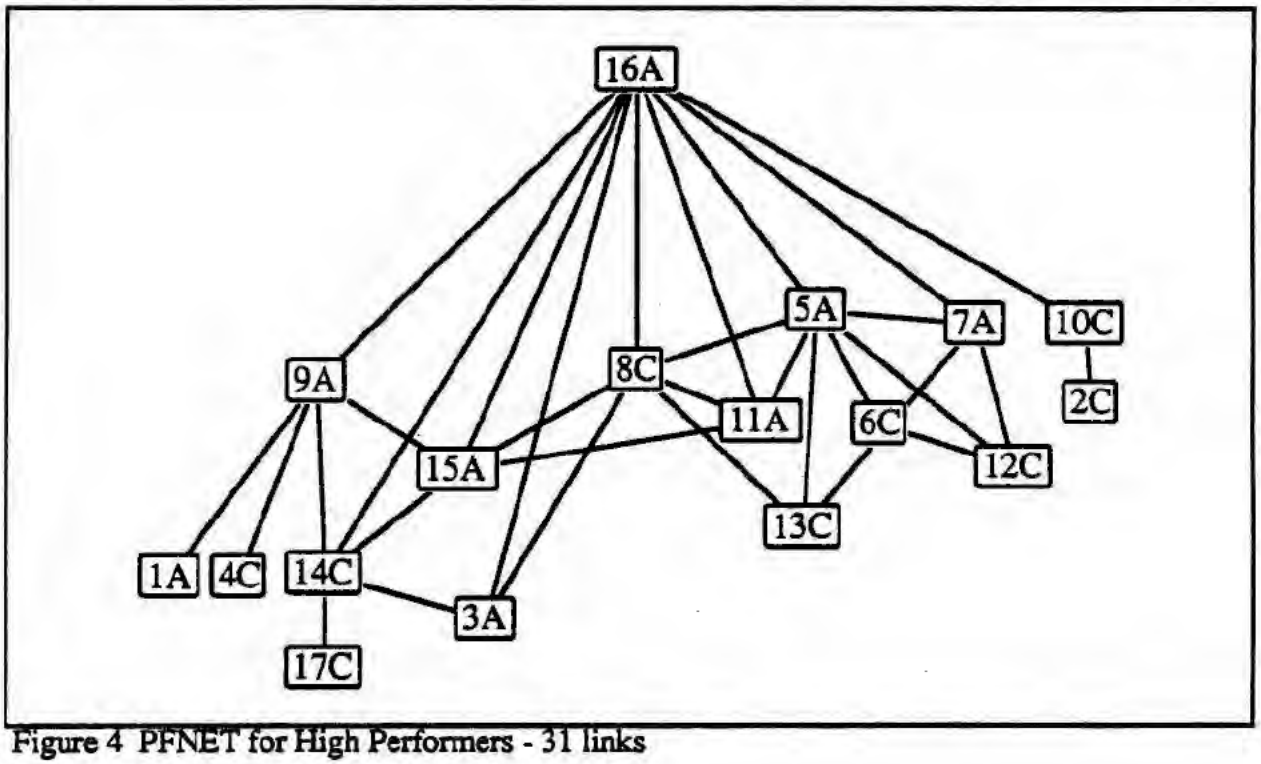


Figure 4 PFNET for High Performers - 31 links

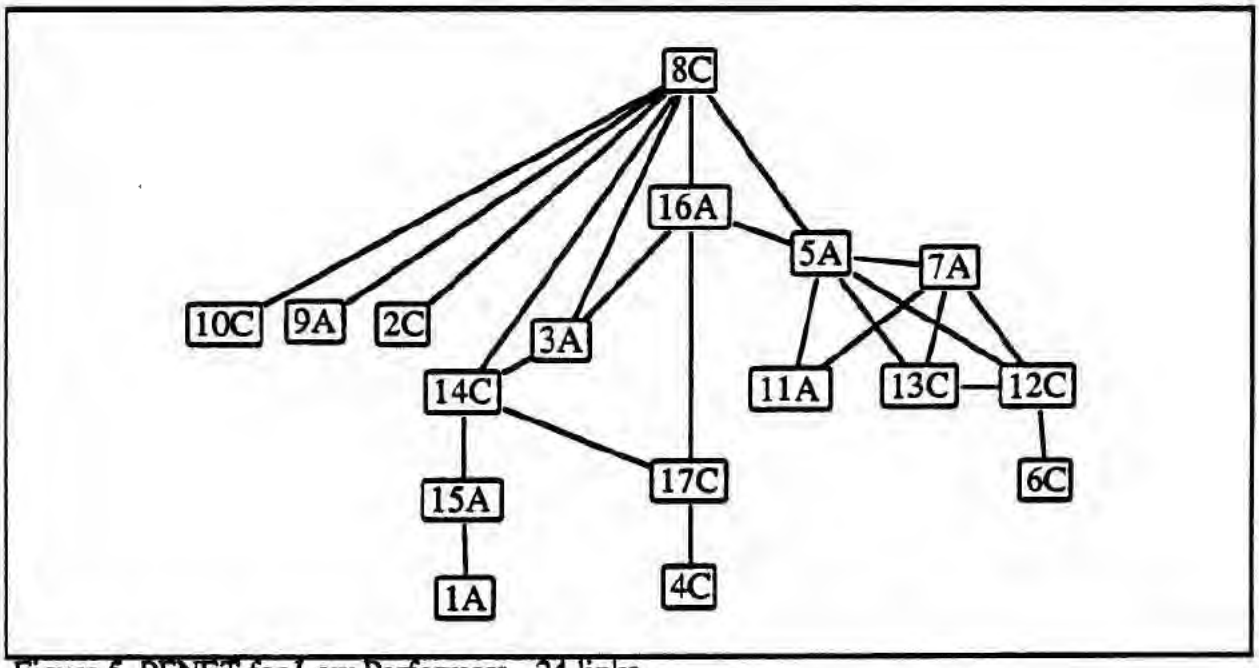


Figure 5 PFNET for Low Performers - 24 links

were dropped from the study. This is consistent with the research methods of Emory (1985) when differences between groups are slight and need to be accentuated. PFNETs were generated for these two groups (Figures 4 and 5) and they were reanalyzed for indications of three of the four characteristics, omitting superior performance.

Holistic perception
In the analysis for holistic perception, the expert group had thirty-one links while the novice group had only twenty-four links. The Wilcoxon Signed Rank Test did reveal a significant difference in the number of these links ($p<.05$, $z=-1.645$). As expected, the expert group has more links than the novice group which is an indication of their more holistic perspective.

Critical thinking
In the analysis for critical thinking, the thirty-two critical thinking scores were regressed against the performance ratings for each group. As discussed previously, an r value of .50 or above is necessary to show significance. The analysis revealed a significant positive correlation ($p<.0001$, $R^2=.72$, $r=.85$) between the performance ratings and the critical thinking scores. For psychological data, this is an unusually strong indication.

Use of abstract concepts
In the analysis for the use of abstract concepts, it was expected that there would be a larger number of links attached to the abstract concepts in the expert PFNET than in the novice PFNET. This would indicate a preference for abstract concepts in the reasoning of the experts. This was in fact the case. The expert PFNET had twenty-nine abstract links to twenty-two abstract links in the novice PFNET. The experts indicated a significantly greater ($p<.0217$, $z=-2.02$) preference for abstract concepts than the novices.

The results of this part of the study clearly indicate that the technicians that demonstrated superior performance at diagnosing machine faults are most likely the true experts since they also demonstrated characteristics that are commonly associated with expertise; holistic perception, critical thinking, and the use of abstract concepts. When classified based on experience (or position), the results were not significant. Only when performance was used as the classification variable were all of the expected results demonstrated.

CONCLUSIONS AND DISCUSSION

The first conclusion of this study is that a rating of observed performance can be an accurate measure of expert diagnostic ability and that this measure can be verified using an independently validated instrument. In this study, the performance measures were strongly correlated with the Critical Thinking Appraisal scores, indicating the presence of superior cognitive ability in the superior performers and lesser ability in the lesser performers. The PFNETs that resulted from classifying the subjects by these performance ratings displayed characteristics that are appropriate for expert and novice differences as supported by prior research.

The second conclusion is that the Pathfinder technique is a viable method for the collection and representation of mental models. The Pathfinder technique has proven to be versatile in domains as diverse as flying, programming, and now diagnostics. It contributes to the study of human cognition as it provides numeric detail representing the specific relationships between concepts. This detail is not available from other commonly used techniques such as Multidimensional Scaling and Cluster Analysis. However, this study has indicated that the Pathfinder technique must be used with proper assumptions.

The primary conclusion of this study is that research in the area of knowledge-based systems, especially knowledge acquisition methods and tools, must provide some means of validating the basic assumptions of the research design. Research in the development of a method or tool that is useful in eliciting expert knowledge cannot be developed from the input of individuals

that lack the cognitive ability to map their experiences into useable structures, no matter how much experience they have had. In this study, it was clearly shown that experienced technicians may or may not possess expert mental abilities. Invalid assumptions concerning the presence of expertise may easily invalidate any conclusions that could have been made from indications within the research data. A review of the literature indicates that this mistake is not uncommon. Consider the scenario where a computer system is being developed based on the knowledge of an individual that was experienced but lacked the mental skills to use those experiences to increase the quality of his problem solving. The resulting system would be a knowledge-based system, but would not be an expert system.

The conclusions of this study introduce several interesting questions. How many expert systems have been developed in the past that are based on the "expertise" of a relative novice? Can this mental ability be taught? What specific modifications should be made to training curricula to increase this ability? Is the Watson-Glaser Critical Thinking Appraisal sufficiently sensitive to measure this increase? How can the Pathfinder technique be used to assist in teaching mental skills?

Several problems were encountered with the implementation of the Pathfinder technique that must still be addressed. The number of concepts that can be used is severely limited unless some method is developed to reduce the number of concept pairs. In this study, it would have been revealing to have included a larger set of concepts. However, this was not practical since only fifty concepts would generate a questionnaire with 1225 responses! How long can a subject maintain the mental focus required by this type of data gathering? There was concern in this study that the subjects would not maintain the required mental focus over the 136 responses which is a much smaller number than was used in the original study. Were the last responses as meaningful as the first responses? In other words, Pathfinder's practical application in areas other than psychological and cognitive science research is questionable if the domain knowledge is too complex or too broad. These problems should be addressed and implementation methodologies developed so that Pathfinder can quickly reach its potential as a cognitive modeling tool.

More research is needed in varied domains (eg. medicine, management, law, engineering) and problem-formulation paradigms (eg. diagnostics, design, control, planning) to identify the appropriate expert characteristics and methodologies for each situation. Performance rating schemes need to be developed that provide greater sensitivity to the variations in a person's level of performance. A method for validation of the ratings schemes should also be developed. Critical thinking is only one of many measures of cognitive ability. Although it was effective in this particular research situation, more data is needed before general applications can be made. Does it provide as strong a measure with other subjects or in other domains? A definition of expertise is needed that goes beyond a recitation of observable qualities. What is expertise in terms of cognitive ability? What specific abilities are involved? Does each domain have its own definition?

Assuming the availability of a practical implementation methodology, the PFNET of the expert would provide valuable information for the knowledge engineer. Expertise, rather than just a set of related facts and rules, would be indicated when the knowledge base and the PFNET of the expert contain similar structural characteristics.

If knowledge engineers are familiar with the structural characteristics of the expertise in a particular problem domain, then they could test for these characteristics in the PFNET of the knowledge source. In an area such as domain definition where few structured techniques exist (Woodward, 1990), a PFNET of the expert would be a valuable indicator of the basic organization of the domain knowledge as well as the structural refinements that are responsible for superior performance. The PFNET could be useful as a tool to organize the knowledge into segments for acquisition. This blueprint would be an invaluable guide to plan the number

and sequence of knowledge acquisition sessions that would be required and the proper methodology to use in each session.

With multiple experts, the differences in the individual PFNETs would provide indications of the individual differences of the experts. Knowledge engineers could also develop specific knowledge acquisition methodologies that will be more effective in the elicitation of expertise (Cooke and McDonald, 1988; Keravnou and Johnson, 1986). If the basic structure of the expertise was known, then the most effective means of interacting with the expert could be devised. New knowledge acquisition methodologies could also contain documentation schemes that would be appropriate for the knowledge structures in that domain and paradigm.

At present, elicitation of anything more than basic domain concepts is a tedious and many times impractical task considering the limited availability of the expert and the volume of detailed knowledge that is required. However, Pathfinder methodologies should make a significant contribution to these tasks as well as reveal as yet undiscovered qualities of that elusive phenomenon we call expertise.

We strongly recommend the use of a validation measure to provide an independent indication of the level of expertise of the individual knowledge sources. Since practitioners are aware of the lack of expertise in many of the systems that are built using expert systems tools (Keyes, 1990), the researcher should also be concerned. Just because the knowledge-based system provides responses that emulate the original knowledge sources is not necessarily an indication that expertise has been captured. Measures from other disciplines, such as Cognitive Psychology, should be investigated for their usefulness in this most crucial task.

REFERENCES

Bouwman, M., (1982), The use of accounting information: expert versus novice behavior. G. Ungson & D. Braunstein (eds.). Decision Making: An Interdisciplinary Inquiry. Boston: Kent.

Brehmer, B., (1980), In one word: not from experience, Acta Psychologica, 45:223-241.

Clark, C. T. & Schkade, L. L., (1979), Statistical Analysis for Administrative Decisions. 3rd ed. Cincinnati: South-Western.

Cooke, N. M. & McDonald, J. E., (1988), The application of psychological scaling techniques to knowledge elicitation for knowledge-based systems. J. Boose & B. Gaines (Eds.). Knowledge-Based Systems. Vol. 2. New York: Academic Press.

Cooke, N. & Schvaneveldt, R., (1988), Effects of computer programming experience on network representations of abstract programming concepts, International Journal of Man-Machine Studies, 29:407-427.

Emory, C. W., (1985b), Business Research Methods. Homewood: Irwin.

Ettenson, R., Shanteau, J., & Krogstad, J., (1987), Expert judgement: is more information better?, Psychological Reports, 60:227-238.

Freyd, J., (1983), Dynamic mental representations and apparent accelerated motion, Dissertation Abstracts International, 44(9):92.

Glaser, R., (1984), Education and thinking, the role of knowledge, American Psychologist, 39(2):93-104.

Jaeger, R. M. & Freijo, T. D. (1975). Race and sex as concomitants of composite halo in teachers' evaluative ratings of pupils. Journal of Educational Psychology. 67(2):226-237.

Johnson, P., Zualkernan, I., & Garber, S., (1987), Specification of expertise, International Journal of Man-Machine Studies, 26:161-181.

Johnson, P., (1983), What kind of expert should a system be?, The Journal of Medicine and Philosophy, 8:77-97.

Kahney, H., (1983), Problem solving by novice programmers. T. Green, S. Payne, & G. Van der Veer (Eds.). Psychology of Computer Use. London: Academic Press.

Keravnou, E. T. & Johnson, L., (1986). Competent Expert Systems: A Study in Fault Diagnosis. NY: McGraw-Hill.
Keyes, J. (1990). Where's the "expert" in expert systems?, AI Expert, March, p. 61-64.
McGraw, K. & Harbison-Briggs, K. A., (1989). Knowledge Acquisition: Principles and Guidelines. N. J.: Prentice-Hall.
Olson, J. & Rueter, H., (1987), Extracting expertise from experts: methods for knowledge acquisition, Expert Systems, 4(3):152.
Schvaneveldt, R., Dearholt, D., & Durso, F., (1988), Graph theoretic foundations of pathfinder networks, Computer Math Applications, 15(4):337-345.
Schvaneveldt, R., Durso, F., Goldsmith, T., Breen, T., Cooke, N., Tucker, R., & DeMaio, J., (1985), Measuring the structure of expertise, International Journal of Man-Machine Studies, 23:699-728.
Simon, A. & Ward, L. (1974). The performance on the Watson-Glaser Critical Thinking Appraisal of university students classified according to sex, type of course pursued, and personality score category, Educational and Psychological Measurement, 34:957-960.
Tenney, Y. & Kurland, L., (1988), The development of troubleshooting expertise in radar mechanics. J. Postka, L. Massey & S. Mutter (Eds.). Intelligent Tutoring Systems: Lessons Learned. Hillsdale:Erlbaum.
Tversky, A. & Kahneman, D., (1983), Extensional versus intuitive reasoning: the conjunctional fallacy in probability judgement, Psychological Review, 90(4):293.
Van Gigch, J. P., (1978), Applied General Systems Theory. NY:Harper & Row.
Watson, G. & Glaser, E. M. (1980). Critical Thinking Appraisal Manual. Harcourt Brace Jovanovich, Inc.
Woodward, B. (1990). Knowledge acquisition at the front end: defining the domain, Knowledge Acquisition, 2:73-94.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Knowledge Acquisition During Interactive Planning: Exploiting Human-Generated Exceptions[1]

Carol A. Broverman and W. Bruce Croft

Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003
email: *broverman@cs.umass.edu*

Abstract

In realistic settings, knowledge bases are inherently incomplete and/or incorrect. In this paper, we advocate the exploitation of inevitable failure points during interactive planning to set up "conduits" through which additional knowledge acquisition can take place. We show how apparent failures caused by the actions of human agents (*exceptions*) can be interpreted constructively as valid events. The SPANDEX system uses a theory of plausible *rationales* to relate unexpected events to valid knowledge about the domain. New knowledge acquired in this fashion refines the knowledge base and planning context, resulting in improved performance.

1 INTRODUCTION

Planners can be used as intelligent assistants to automate or support a variety of complex tasks [LC89, Sac77, Wil88, CL84]. Most planning research, however, has been done in very simple domains, using autonomous non-human agents (e.g., robots) to execute the generated plan. In realistic settings, however, several factors limit the utility of such knowledge-based assistance: e.g., the need for human supervision, the open-endedness of complex and dynamic domains, and the frequency of unexpected occurrences which can upset a calculated plan. In this paper, we argue that in many cases humans should remain in the system loop, and that unexpected plan disturbances related to their actions can be used as a focal point for incremental refinement and extension of the domain model.

First, we claim that the assumption that an autonomous non-rational agent will be the executor of a course of action produced by a planner is too restrictive. The need to relax the assumption of agent automony is supported by recent trends in planning research. In many settings total automony is not desired, and the human agents involved must still retain a large measure of control. The reason for such a configuration may involve the measure of risk involved; mistakes may be costly and the human must always be the one to make the final decisions. Examples are the use of a planner to guide a fighter pilot, or to aid a physician in prescribing a treatment plan for a patient. In addition, planning techniques can be used to support intelligent task development and monitoring, used as a form of assistance under the control of a human planner. An example of this usage is to help a conference chair plan an academic conference. In all such systems, planning takes place in an interactive and supervised fashion, with execution interleaved with planning actions.

[1]This research was supported in part by a contract with Triumph-Adler AG and Ing. C. Olivetti and by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, NY 13441-5700.

Static knowledge bases in realistic domains are inherently incomplete and incorrect. This fact, coupled with the unpredictability of both human agents and the environment suggests that systems need the capability to respond intelligently to unexpected changes that induce system failure. Recovery measures such as those of [AIS88, Hay75, Wil88] and other transformational approaches [SD87, Sim88, RDF85] have been proposed to effectively replan around arbitrary unanticipated domain state changes. However, we consider the unique role of human agents, and advocate a constructive approach that can extend the boundaries of the initial knowledge base.

We claim that events that initially appear to be points of failure may instead represent valid domain information that should be acquired by the system. For example, in conference planning, when the program chair for a conference selects a reviewer who is not known to be qualified in the area of a submission, it may be the case that the model is missing key information stating that the individual is indeed an expert in the area. Alternatively, a conference organizer can opt for unanticipated shortcuts during the planning of a conference, e.g. by invoking a sponsor contract service to issue the call for papers instead of arranging the typesetting, printing and distribution of the call on his/her own.

The natural intelligence and familiarity of humans with the application domain implies that their actions, such as those just described, are generally purposeful even when they may be inconsistent with system expectations. Human-generated exceptional occurrences should be incorporated as contributory events whenever possible, rather than "undone" using traditional replanning techniques.

The fundamental idea behind our approach is to conduct knowledge acquisition opportunistically during the empirical use of the system. When an exceptional event occurs, it is viewed as an opportunity to refine the domain model, and a "conduit" is established for the exchange and incorporation of information to justify the exception. The domain model, a theory of human procedural behavior, and information elicited from the user, are applied to transform the current plan into a valid alternative. A primary goal is to provide an explanation which will allow the continuation of planning and execution [BC87, BC88]. As an important side-effect, the explanation process results in new domain knowledge about how tasks can be performed in the domain and the domain objects which are manipulated. By taking this "constructive" approach during exception handling, two important goals are attained: domain knowledge is incrementally refined, and replanning can be avoided or reduced.

In this paper, we describe our methodology for handling points of failure, which we call *exceptions*, as they arise during interactive planning, and show how it can result in an augmented knowledge base. First, we present a theory of human procedural behavior which we use as the basis for understanding how exceptional agent behavior during plan execution can represent new knowledge about the domain. We then describe a framework for interactive planning, and define the types of exceptions that can be detected within this framework. A general architecture is presented which has been designed and implemented to refine and extend the knowledge base within the context of exception handling. We show how the plausible *rationales* identified by our theory are used as the basis for explanations of exceptions. Corrective measures are proposed in the form of *amendments*, which represent new knowledge to incorporate into the static and dynamic models of the domain. In the final section, we describe the implementation of these ideas in the SPANDEX system using an example from the domain of conference planning.

2 A Theory of Human Procedural Behavior

One of the major goals of this work is to address the human component in plan execution and plan revision. We believe that procedural deviations performed by human agents can represent new and valid knowledge that can be acquired about the domain. To provide a model for this claim, we have adopted a theory of human procedural behavior that is closely tied to research analyzing deviations of human agents from typical plans of action. We base this theory on data made available through studies identifying aberrations that occur when humans perform procedural tasks [RM82, Nor81, Ras80, Hol87].

We adopt a unique perspective of what many have termed as "errors" when syntactically detecting unanticipated deviations in a plan monitoring context. This perspective is predicated on two key assumptions: (1) A computer-generated plan is often based on an inherently incomplete model of the domain; and (2) Human agents behave intelligently and purposefully. In the domains in which human errors have been studied, the violated procedures are often fixed and the procedural knowledge is bounded. Therefore, a deviation from a plan of action in these settings is almost always regarded as an incorrect action. However, when an inherently incomplete domain model is used to generate the plan of action, the identical syntactic deviation detected during plan execution may in fact represent an intentional and valid step towards the pursuit of the overall plan goals. The recognition of this action as valid, through explanation, can refine the incomplete model of the domain.

Although taxonomies of human procedural deviations generally refer to such aberrations as "errors," we prefer to use the term *exceptions* to avoid an overriding negative connotation. As already suggested, some of these exceptions may be purposeful and valid, especially in the context of an underspecified domain model. Also, it is important to distinguish between the surface manifestation of an unexpected action during the execution of a procedure and the intent which may have generated the action. In general, the taxonomies we examined intermix these two aspects of exceptions, although Hollnagel does underscore the importance of distinguishing between the syntactic appearance of an exception (*phenotype*) from its underlying cause (*genotype*) [Hol87].

Most of the exceptions identified by established taxonomies fall into one of four general categories: repetition, use of wrong object, intrusion, and omission. Particular examples of these categories are: performing actions out-of-order, skipping steps, reversing steps (known as behavioral spoonerisms), exchanging resources with similar functions, exchanging resources with physical similarities, inserting extra actions, and repeating actions [RM82]. We hypothesize constructive intentions motivating these actions, and have derived a set of *plausible rationales* to represent these intentions. The plausible rationales are then used as the basis of explanation when considering the role of an unexpected event in the current plan context. The rationales can be grouped into the following categories:

1. **Knows alternative.** The agent is performing an action which is an alternative way to achieve an action anticipated by the planner.
2. **Knows shortcut.** The agent knows a way to skip some of the anticipated steps of the plan and still achieve the high-level goal. Alternatively, the agent is able to accomplish an abstract plan goal in "one fell swoop."
3. **Can reorder.** The agent is able to perform the anticipated plan steps but in an alternative order than that predicted by the planner. The original ordering may be overconstrained.

4. **Knows reordering shortcut.** The agent is able to perform an action which accomplishes a number of plan steps that are anticipated later in the plan. (This is a combination of the previous two rationales.)

5. **Introduces new action.** The agent is introducing a new action that is not anticipated by the planner but is part of achieving the high-level goal. The original activity description may be incomplete.

6. **Introduces redundant action.** The agent is repeating a plan step that has already been performed.

7. **Introduces extraneous action.** The agent is introducing a new action that is not anticipated by the planner, nor is part of achieving the high-level goal, but does not harm the ongoing plan in any way.

8. **Use alternative resource.** The agent is performing an action that is using a resource similar to the resource the planner expected to use.

9. **Normal action in an imperfect domain.** For completeness, we include the case in which the agent is not varying the course of action in any way.

3 The Backdrop: Planning, Execution and Exceptions

In this section, we give a formalization of interactive planning and exceptions, which is implemented in the POLYMER planning system [LC89]. We also identify potential entry points within this framework for the acquisition of new knowledge about the domain.

3.1 A Model of Planning and Execution

A planning problem is specified as: (1) an initial world state s_i, (2) a desired final goal state s_f, and (3) a set of domain activities A which can be applied in some order to s_i to produce s_f. A planner iteratively elaborates the top-level goal representing the final goal specification s_f. The result of each of these manipulations is represented by a *plan network*, which is viewed as the current version of an evolving plan. A plan network is a strict partial order and consists of the following elements:

- N: a set of nodes, where each node is either a *goal* node (representing a decomposable activity), or an *executable* node (representing a primitive action that can be executed by a tool or human agent);
- L: a set of temporal links which establish a partial ordering among the nodes;
- W: a set of world states, which are snapshots of the dynamic domain model. Two of these world states are attached to each node in N to describe the world states believed by the planner to hold before *(before-world)* and after *(after-world)* the execution (for an executable node) or achievement (for a goal node) of that node. Each node also contains a set of conditions C that must hold in the before-world of that node before it can be processed by the planner;
- I: a set of *protection intervals*, where each interval specification designates a partial world state and the temporal range during which it must be maintained.

We further define a set of criteria which must hold in order for a plan network to be consistent:

- All before-worlds and after-worlds in W are internally consistent with respect to constraints in the domain model.
- All activity preconditions (a subset of C) specified by nodes in the current plan must be satisfied in the before-worlds of their respective nodes.
- Constraints imposed on resources used by an activity (a subset of C) must be satisfied in the before-world of each node in an expansion of that activity.
- The after-world of each plan node must be consistent with the goal of that node.
- All protection intervals in I must hold.
- The set of temporal ordering specifications L must be consistent.

We define the process of *planning* as iterative *transformations* on plan networks. Since planning is interleaved with execution, we adopt a broad definition of planning that subsumes execution. Therefore, we define a *final plan* as a plan network which has been fully ordered, and every node is either a *phantom*[2] or it is a primitive activity node that has been executed. Thus, a plan network represents a class of final plans; many different possible final plans can result depending on the subsequent choices of elaborations and operations. Conceptually, a new plan network results each time an operation is performed by the plan network maintenance system (PNMS [BL89]), such as node expansion, node execution, node ordering, or protection interval establishment. The complete *plan history* is a lattice containing all intermediate plan networks ordered within planning time, where the distinguished upper bound is the eventual final plan. The relation is a partial order since backtracking may be allowed. The plan history maintains a record of all planning actions performed.

Plan networks can also be described in terms of the stage of their execution. Since the domains we are concerned with generally interleave planning with execution, plan networks are often partially executed. Associated with each plan network is a set of *expected action nodes* which are eligible for execution. A node is in this set if and only if: (1) it is a primitive activity node, (2) all of the conditions C specified by the node are satisfied in the before-world of that node, and (3) all of the node's necessary predecessors are complete and awaiting successors[3]. When such a primitive activity node is selected for processing, a request is issued to the human agent or tool responsible for that action.

3.2 Exceptions

Exceptions during planning (including execution) are signalled by the detection of one or more of a defined set of inconsistencies. In this section, we outline the types of possible inconsistencies, and in later sections we show how the intelligent handling of these inconsistencies can result in an improved and extended domain model.

There are two classes of inconsistencies that trigger the exception handler. The first class is detected during interaction with the user, when planner expectations are contradicted by input from the user. Even though the user input may represent valid domain information, an initial discrepancy results.

[2] A phantom node is a goal node which has been determined to be true at its position in the plan without further expansion and execution [Wil88].

[3] A complete definition of expected action nodes can be found in [LC89].

The second class of detectable inconsistencies results from planner actions and is independent of user interaction. Both types of inconsistencies may represent error or gaps in the initial domain model.

Exceptions During User Interaction. In POLYMER, user interaction occurs at four specified points during interactive planning. The first two of these occur when the user is called upon to help the planner disambiguate at choice points, and the second two interaction points occur during the execution of primitive activities.

- **The user chooses from possible activities.** The user can choose from a set of possible activities to pursue during goal expansion. For example, when developing a plan for staging a conference, the planner may find two possible activities for making hotel arrangements for a conference: contract-out-hotel-arrangements or choose-and-book-hotel-yourself. The user is asked to disambiguate. Two types of exceptions are possible here:
  - no-activity-possible: The planner may be unable to find any existing activities which can achieve the posted goal. This is viewed initially as a failure point since the planner is unable to continue without additional information.
  - activity-choice-inconsistent-with-expectations: The user may elect to choose an activity to use for planning that is not among the activities presented which are known to he capable of accomplishing this goal.
- **The user makes resource choices.** The user must choose bindings for necessary resources when an activity is selected to achieve a posted goal. For example, a particular program.chair must be chosen before conference submissions can be received and reviewed. The user is presented with a menu of choices that are known to satisfy the current constraints. The user is asked to disambiguate, if multiple resource bindings are possible. Types of exceptions which may result are:
  - no-known-resource-available: No objects are available in the knowledge base which satisfy the current constraints on this resource.
  - resource-choice-inconsistent-with-expectations: The user may elect to choose a resource binding that is not among the suggested bindings which are known to satisfy the current constraints.
- **The user performs an agent-executable action.** When the user is asked to perform an agent-executable activity to achieve a goal, they are asked to respond either with the appropriate verification or an indication that they have proceeded in an alternative fashion.
- **The user is asked to verify the invocation of a tool action.** When the system is about to invoke a tool to perform a tool-executable action, verification is elicited from the user. Again, the user can override the intended action of the planner at this point.

  The exception that may result in either of the above cases:
  - agent-response-inconsistent-with-executable-expectation: the agent does not verify the agent action or tool action proposed by the planner; instead chooses to do something different.

Exceptions Resulting From Planner Operations. The second class of potential inconsistencies are those pertaining to the state of the current plan network. The plan network may now be inconsistent, either as a result of (a) planning actions which do not involve the user, or (b) the incorporation of what appeared to be a valid user action. Specifically, one or more of the plan network consistency criteria defined in Section 3.1 may have been violated. Two additional exception types result:

- plan-network-inconsistent-during-planning: inconsistencies result after a planning action in which an agent is not directly involved, e.g., during straightforward plan expansion;
- plan-network-inconsistent-after-execution: execution by agent proceeds normally, but after asserting the goal and effects of the monitored agent action, inconsistencies result.

All of the exceptions outlined above can represent errors or gaps in the domain model, and can be used as focal points for the acquisition of new domain knowledge. Note that exceptions in the first class are detected before the results of the interaction are reflected in the domain and plan model, so handling them a priori can avoid additional inconsistencies which would result from a simple insertion of the agent response. Exceptions which fall into the second class are not detected during interaction with the user, but rather when the incorporation of a seemingly valid planner or agent action into the global plan context results in a violation.

The Exception Problem. We are now in a position to formulate the actual *exception problem*. The current state of the plan is represented as a plan network p with an identified set of expected action nodes. A planning action is taken, either to elaborate the plan further, or to invoke an agent or tool for execution. If inconsistencies result, the SPANDEX system is invoked as follows:

- **Given:** (1) p: a partially executed plan network; (2) D: a model of the domain, and (3) X: a set of calculated inconsistencies;
- **Compute:** (1) p': a new successor plan network which meets the following criteria: (a) The set of executed nodes in p' include all executed nodes in p; (b) p' is consistent, and (c) p' has the same top-level goal as p; and (2): D': a refined or expanded model of the domain.

Our general approach to this problem is to manipulate available domain knowledge to generate plausible explanations indicating how the current network and domain model can be transformed to eliminate the calculated inconsistencies. The explanation embodies new information about alternative ways to accomplish tasks in the current domain. In addition, the selection of an explanation may involve the acquisition of additional information from the user. Successful handling of the exception results in a refinement of the domain model.

4 System Architecture

The architecture of the SPANDEX system designed to handle this problem is shown in Figure 1. When an exception is detected during planning or by the plan execution monitor, the *exception classifier* is invoked to compute the membership of the exception within the predetermined set of exception classes, as described in Section 3.2. Further classification may be necessary in order

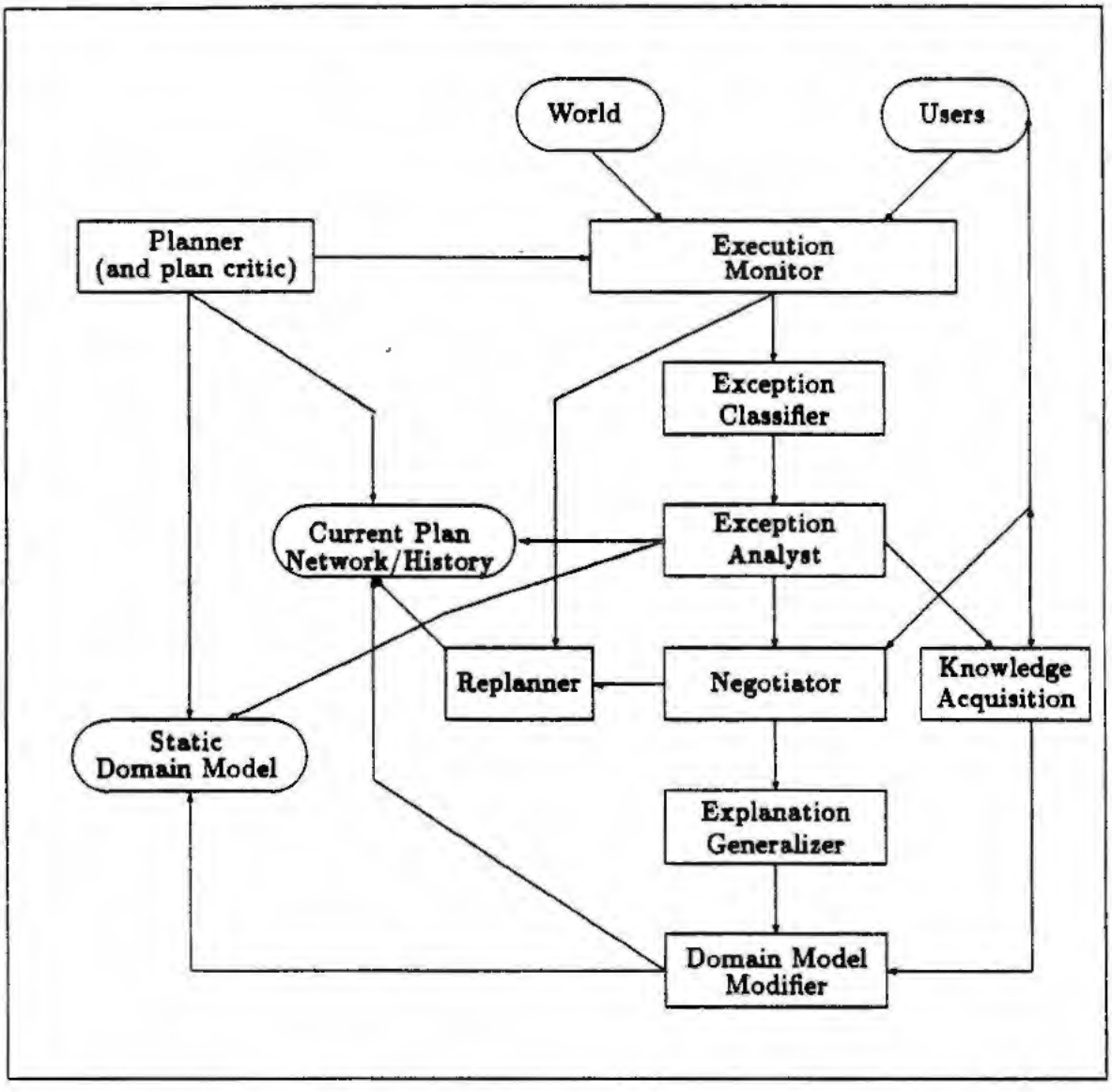


Figure 1: An architecture for planning and exception handling

to determine particular characteristics of how the agent's actions may relate to the current plan context. The *plan critic* calculates the set of inconsistencies which are present, such as the violation of achieved or protected goal states in the plan. The *replanner* handles *unaccountable* exceptions (generated by unknown agents who are represented by *world* in Figure 1) using plan repair methods based on those of [Wil88].

Knowledge acquisition takes place explicitly upon the occurrence of the no-activity-found and no-known-resource-available exceptions. A graphical interface [MSC89] is invoked to acquire the description of a new activity (to satisfy the pending goal) or a new object (to use as the required resource). The *domain model modifier* then incorporates this new description into the existing knowledge base. When a new activity description is acquired, the planner is provided with an additional potential way to achieve the goal of other subsequent subtasks. When a new domain resource object is defined, additional resource bindings may now be available for future plan steps. After

the knowledge acquisition session has been completed, control is passed to the *exception analyst*.

The *exception analyst* module applies domain knowledge according to a set of prespecified plausible *rationales* to construct explanations of *accountable* exceptions (generated by known agents and represented by *users* in Figure 1). The rationales are based on the model of human procedural behavior described in Section 2, and the mechanisms of the exception analyst which use these rationales to produce explanations are described in Section 5.

Since several agents can be affected by an exception, we propose the use of negotiation [Fik82, SMR86] to establish a consensus among them regarding the explanation and proposed plan modifications. The *negotiator* identifies the affected agents and uses the information provided by the exception analyst to conduct a dialogue. The output of the negotiation phase is a selected explanation, along with approved changes or *amendments* to be made to either the plan network instantiation or to the static domain knowledge base.

The *explanation generalizer* produces a generalized form of the verified explanation, using taxonomic information in the domain model. This new knowledge about domain activities and the amendments resulting from negotiation are passed to the *domain model modifier* for implementation. Thus, a successful negotiation can result in a system which has "learned," that is, the static domain model may be augmented with knowledge about the exception and thus the system is able to handle future similar exceptions.

5 Explanation Generation Using Rationales

In this section, we present the *exception analyst* component of our general architecture in more detail, and discuss how the resulting explanations can add to the domain model.

Explanations of an exceptional action are generated by the controlled application of a set of *plausible inference rules* (PIs). An example PI is shown in Figure 2. In simpler terms, this PI states that when an inconsistent agent response is detected while executing a primitive activity, and the type of action which was accomplished is a specialization of the type of action that was expected by the planner, the accomplished action can be considered to be an alternative way to achieve the pending goal. For example, this might be an appropriate line of reasoning to pursue in the conference planning domain if a program chair rejects a submission immediately because of an excessive word count, rather than initiating the review process. Both actions achieve the goal of reaching a decision on the submission. In such a case, a representation of the predicted action can be replaced by a node representing the accomplished action.

Each PI maps from a state specification S to an explanation E. The specification of S consists of a set of predefined indicators and inconsistencies which are specified as: (1) relations between objects in the domain model, (2) status of execution monitor state variables, or (3) qualifications on the plan network state. The explanation E also has two components: a *rationale*, and an *amendment*. The rationale gives a semantic basis for the exception, suggesting its contribution to the ongoing plan. The set of rationales which are currently used by the system are described in Section 2. Each rationale has associated strategies which dictate how to look for corroborating evidence (as described by the state specification S). If found, evidence provides a measure of support for the rationale in justifying for the exceptional behavior.

An amendment prescribes the changes needed to establish the rationale and restore system consis-

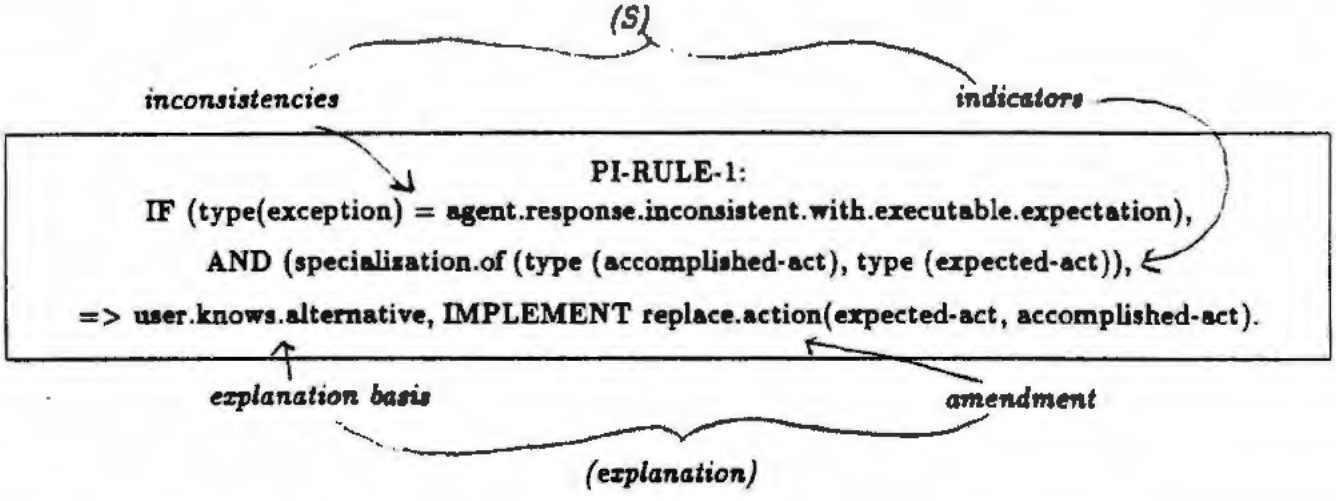


Figure 2: An example plausible inference rule (PI)

tency. It consists of one or more plan network alterations and primitive modifications of the domain model. Examples of plan network alterations include: (1) replacing one of the expected actions with a node representing the unexpected action, (2) replacing a wedge[4] containing one of the expected actions with a node representing the unexpected action, and (3) replacing a later node with a node representing the unexpected action and deleting the intervening nodes. The plan network alterations are composed of the primitive plan network operations delete-node, insert-node, expand-node, and establish-ordering. Possible domain model modifications are the addition or deletion of values to a field of an object, the insertion or removal of a taxonomic link, or the modification of a constraint.

When amendments are necessary to resolve a violated constraint, there are two fundamental approaches to resolving the problem[5]:

1. Relax the constraint. Possible ways to do this are:
   (a) Disjunct addition (add a disjunct to the specification);
   (b) Conjunct elimination (eliminate one or more conjuncts from the specification);
   (c) Taxonomic generalization (replace a class specification with a more general superclass);
   (d) Taxonomic expansion (replace a class specification by the set of classes which are subsumed by the original class specification);
   (e) Range extension (extend a numeric or other ordinal range);
   (f) Constraint elimination (eliminate the constraint);
2. Remove the source of constraint violation.

[4] We define the concept of a *plan wedge* as a node in plan history along with its descendants (determined recursively) which have been introduced through one or more levels of plan expansion. This definition is based on that used by Wilkins [Wil88] and Sacerdoti [Sac77].

[5] The first approach is largely based on work described in [BW85].

(a) The constraint may have failed because of unknown information. Perhaps the missing knowledge can be acquired, resulting in a successful evaluation of the constraint.

(b) Information in the knowledge base may be incorrect. A change should be made to the knowledge base so that the subsequent evaluation of the constraint is successful.

Conceptually, the first approach implies that a constraint was specified incorrectly; it was too strict. In this case, the specification of the constraint itself is changed to accomodate the exception. The second approach implies that some knowledge is either not explicit or is incorrect in the knowledge base. In such cases, the constraint itself is left untouched, but additions or modifications are made to entities in the domain knowledge base, effectively removing the source of the constraint violation.

Explanations can be either *complete* or *incomplete*. A complete explanation results from the application of a PI rule whose state specification S holds completely in the current world model, while an incomplete explanation results from a PI rule having one or more unconfirmed indicators in its state specification. Since we are interested in adding to an inherently incomplete domain model, we consider the validation of incomplete explanations. An attempt is made to confirm missing indicators through interactive dialogue, producing additional plausible explanations while adding to the domain model.

In order to intelligently control the application of PI's and the presentation and selection of the resulting explanations, we use a set of heuristics similar to those applied to plan recognition problems [CLM84]:

- **Completeness**: Prefer a plausible inference rule with more confirmed components in its state specification S.
- **Locality**: Prefer a plausible inference rule that considers an expected action (or wedge) to one considering a later action (or wedge).
- **Cost**: Prefer a plausible inference rule that proposes fewer modifications in its amendment specification to one proposing more modifications.

A threshold is set to limit the number of explanations produced. The most likely explanations are presented to the user in an interactive fashion and a choice is requested. If none of these explanations is acceptable, the process is iterated and the next set of explanations are produced and presented, until an explanation is selected. If no explanation is selected, SPANDEX treats the exception as a true error, invoking general replanning measures to counteract the negative effects. If an explanation is selected, the amendments are applied, and SPANDEX checks the resulting network for consistency. Any remaining violations are handled by the replanner or through an interactive acquisition session with a human agent.

6 EXAMPLE

In this section we present an example from the domain of conference planning. The example illustrates the application of a subset of the defined *rationales* and the opportunities for knowledge acquisition, through (1) explicit invocation (as triggered by a gap in the domain model), (2) the incorporation of a complete explanation, and (3) the completion of an incomplete explanation.

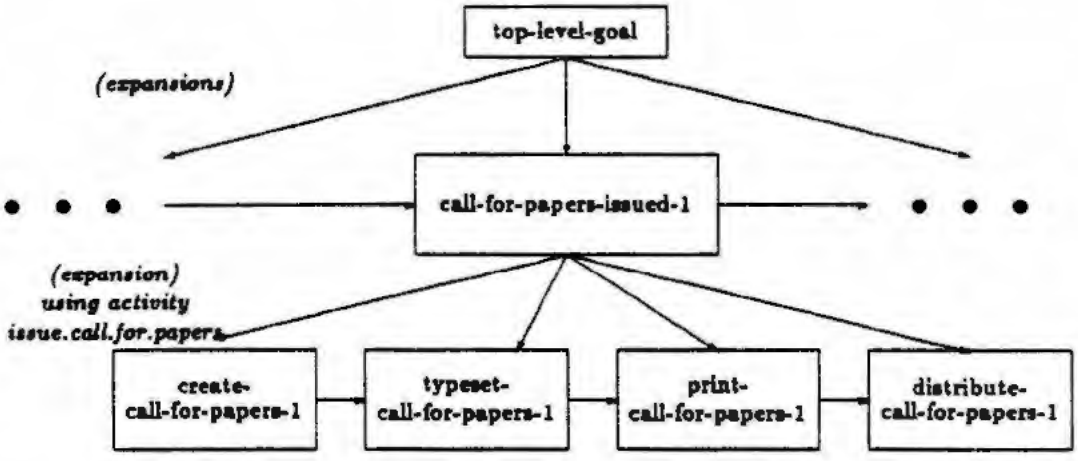


Figure 3: Plan Net for Issuing a Call for Papers

Suppose a conference organizer is setting up an academic conference. This task involves many interrelated steps, such as getting approval to hold the conference, soliciting and refereeing submissions, and arranging logistics such as housing and meals. As part of planning for the conference, a call-for-papers is usually issued by the program chair. The way this is commonly done is to prepare the actual content of the call for papers, have it typeset, then printed, and finally, distributed. Since the knowledge base contains a single activity issue-call-for-papers which embodies this task knowledge, the planner selects that activity to achieve the goal call-for-papers-issued. Figure 3 shows the plan network after the activity selection.

The program chair Larry-Lefkowitz-PhD next creates the content of the actual call-for-papers as anticipated. He is then asked by the system to execute the typeset-call-for-papers-1 action. However, the organisation which is sponsoring the conference has recently begun to offer a contract service to handle the various stages of producing and distributing a call-for-papers, and the program chair would like to make use of this new service. Since this is not one of the known possibilities presented to the organizer, Larry-Lefkowitz-PhD responds that he would like to perform another activity instead.

The system recognizes that an exception has occurred (agent-response-inconsistent-with-executable-expectation). An acquisition session is established in order to resolve the exception. First, the system issues a request to Larry-Lefkowitz-PhD to indicate the performed activity from a menu which includes all known activities, as well as a New choice. He selects New, which the system recognizes as a cue to acquire a new activity description from the agent. Larry-Lefkowitz-PhD uses the DACRON graphical interface [Mah90] to specify the new activity contract-out-call-for-papers, as shown in Figure 4. This new description is assimilated into the knowledge base for future use.

The task for the exception analyst is to now determine how this newly specified activity relates to the current model of the task. First, an exception-record is created to summarize the exception (see Figure 5). SPANDEX generates explanations according to stored strategies associated with each of the known rationales which can underlie inconsistent agent responses (as described in Section 2). Each possible resulting explanation for the exception is represented as a rationale-establishment-record.

Two *complete* explanations result from the application of retrieved PI rules; both are valid, but the one chosen interactively through negotiation with the user is shown in Figure 6. The chosen

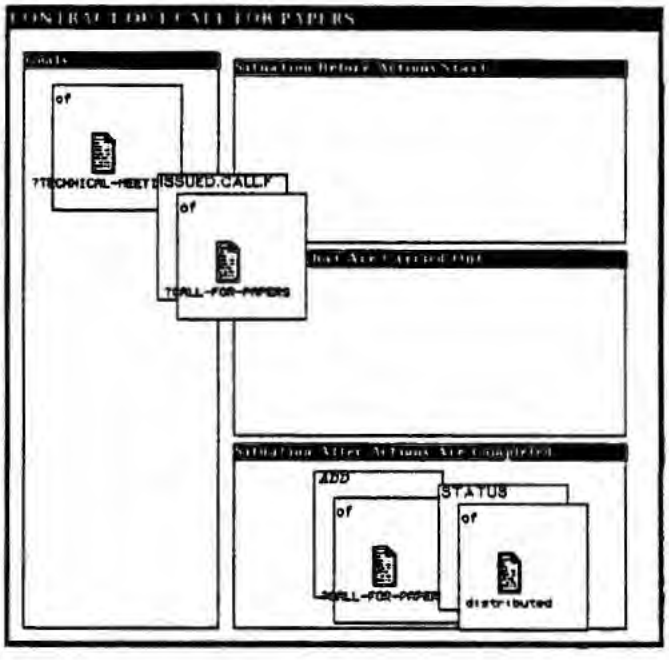


Figure 4: The newly acquired activity CONTRACT-OUT-CALL-FOR-PAPERS

Unit-name: Exception-record-1
Exception-type: agent-response-inconsistent-with-executable-expectation
Unit-comment: "The action performed did not match a planner executable expectation."
Exception-summary:
"The suggested.action: typeset-call-for-papers-1 did not occur;
contract-out-call-for-papers was the performed-action."
Suggested-action:typeset-call-for-papers-1
Accomplished-action contract-out-call-for-papers.
Noted-inconsistencies: agent-response-inconsistent-with-executable-expectation
Possible-rationale-establishment-records: subsumes-current-step-1, alternative-to-current-action-1
Valid-rationale-establishment-records: subsumes-current-step-1, additional-step-to-achieve-current-goal-1

Figure 5: EXCEPTION-RECORD-1

Unit-name: Subsumes-current-step-1
Unit-comment: "The unexpected action is an alternative
to an in-progress-parent-node subsuming an expected action."
Rationale-summary: "The unexpected action contract-out-call-for-papers is sufficient
since its goal matches that of an in-progress activity issue-call-for-papers,
used to achieve in-progress-parent-node call-for-papers-issued-1. "
Status: complete
Comparison-node: call-for-papers-issued-1
Rationale: shortcut
Strategy: subsumes-current-step
Indicators: Equivalent-achieved-goal-and-comparison-goal
In-progress-parent-activity: issue-call-for-papers
Inconsistencies: agent-response-inconsistent-with-executable-expectation
Hierarchy-level-difference: 2
Amendments: Replace-plan-wedge-1

Figure 6: Chosen rationale-establishment-record for EXCEPTION-RECORD-1

Unit-name: Replace-plan-wedge-1
Unit-comment: "Replace a wedge of the plan subsumed by a single node by a new node."
Amendment-summary: "Replace the plan wedge subsumed by
call-for-papers-issued-1 with contract-out-call-for-papers-1."
Implementation: (do (replace-wedge call-for-papers-issued-1
contract-out-call-for-papers-1)
(deactivate typeset-call-for-papers-1 print-call-for-papers-1
distribute-call-for-papers-1))

Figure 7: Amendment record for SUBSUMES-CURRENT-STEP-1

explanation record is based on the shortcut rationale, and states that since the activity contract-out-call-for-papers has been verified to achieve the same goal as the activity issue-call-for-papers, this unexpected action may be a substitution for the abstract goal node which derived the issue-call-for-papers expansion (refer back to Figure 3). In addition, this substitution subsumes the expected action typeset-call-for-papers-1.

An amendment record is next constructed for the explanation which specifies the changes that must be made to the current representation of the plan in order to restore consistency (see Figure 7). The implementation of this explanation involves replacing the wedge of the plan network subsumed by the more abstract parent node (call-for-papers-issued-1) with the unexpected action (contract-out-call-for-papers-1). As a side effect, the unexecuted nodes in the expansion of call-for-papers-issued-1 are deactivated from the planner's predictions. The new plan network which results from the implementation of the amendment just described is shown in Figure 8.

We have shown how the successful handling of an agent-response-inconsistent-with-executable-expectation exception can result in the acquisition of new knowledge and allow continued planning. Now suppose that at a later point in the same conference planning task, Larry-Lefkowitz-PhD is asked to select a particular individual to review one of the submissions received. The paper in question has been received for review in the area of artificial-intelligence. Suppose that a constraint on the reviewer-selected task subgoal specifies that the reviewer selected must have an area of expertise

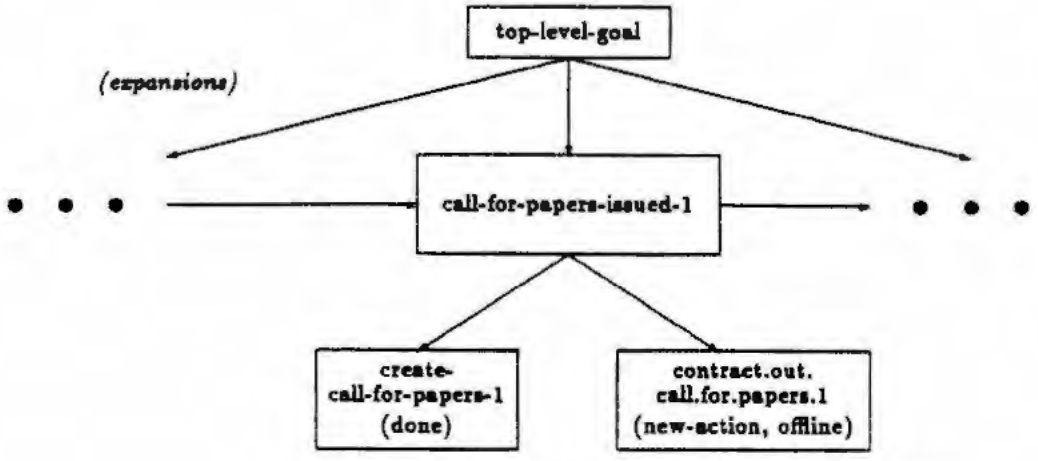


Figure 8: Plan Network for Resolved Exception

which matches the area of the paper to be reviewed.

When the program chair is instructed to perform the activity select-reviewer, those reviewers which are known to satisfy the specified constraints are presented for selection. Larry-Lefkowitz-PhD, however, would like to designate Seymour-Wright as the reviewer of this paper, and although this is not one of the listed choices, Larry-Lefkowitz-PhD chooses Other from the menu to specify the alternate selection. The system makes note of this resource selection, and SPANDEX is invoked upon the detection of this resource-choice-inconsistent-with-expectations exception.

Since the selected reviewer Seymour-Wright is already in the knowledge base, no explicit knowledge acquisition is needed to define a new entity. However, the exception analyst must determine why this choice was not one of the valid suggestions. Constraints on the ?reviewer resource are evaluated, and it is discovered that the area of expertise of Seymour-Wright is computer-vision rather than artificial-intelligence. Thus, a constraint which was dynamically posted on a knowledge base object has been violated, so a further inconsistency dynamic-object-constraint-violation is noted for this exception. The exception-record pictured in Figure 9 summarizes the exception. As with the case of the previous exception, strategies are now invoked to create explanations for the exception.

Two of the plausible explanations constructed by SPANDEX for this exception are shown in Figure 10, one which is complete, and one which is incomplete. In this particular case, the complete explanation was based on the following plausible inference rule (stated informally here):

> **PI-rule-2**: If the violating value is a specialization of the target value of the violated constraint, then the violating value may suffice as a substitution for the target value.

In other words, the complete explanation states that since Seymour-Wright has an area-of-expertise which is a specialization of the required area-of-expertise, he is sufficiently qualified to review the paper of concern. Suppose the user now selects the single complete explanation, which is described in Figure 10. In order to incorporate this explanation into the current context, SPANDEX constructs possible amendments which are shown in Figure 11. Note that the first two proposed amendments illustrate the first approach discussed earlier for handling constraint violations — modifying the violated constraint. They use the techniques of disjunct addition and taxonomic expansion to actually change the constraint specification to accommodate the exception. The third amendment

Unit-name: Exception-record-2
Exception-type: Resource-choice-inconsistent-with-expectations
Unit-comment: "A constraint dynamically posted on a KB object has been violated. "
Expected-action: select-reviewer-1
Accomplished-action: select-reviewer-1
Exception-summary:
"The target constraint: (area-of-expertise Seymour-Wright artificial-intelligence)
was not met; it is the case that (area-of-expertise Seymour-Wright computer-vision)."
Noted-inconsistencies: resource-choice-inconsistent-with-expectations,
dynamic-object-constraint-violation
Target-constraint: (area-of-expertise Seymour-Wright artificial-intelligence)
Perceived-constraint: (area-of-expertise Seymour-Wright computer-vision)
Rationale-establishment-records: specialization-constraint-values-expl-14, generalization-constraint-values-expl-15 ...
Valid-rationale-establishment-records: specialization-constraint-values-expl-14

Figure 9: Exception-record-2

Unit-name: Specialization-constraint-values-expl-14
Unit-comment: "Show that the actual value of the violated constraint predicate is a
specialisation of the target value in the constraint."
Explanation-summary: "The actual value of the area-of-expertise field of Seymour-Wright
(computer-vision) is sufficient since it is a specialisation of the desired value artificial-intelligence."
Status: complete
Rationale: use-alternative-resource
Strategy: specialization-constraint-values
Hierarchy-level-difference: 1

Unit-name: Generalization-constraint-values-expl-15
Unit-comment: "Show that the actual value of the violated constraint predicate is a
generalisation of the target value in the constraint."
Explanation-summary: "If the value of the area-of-expertise field of Seymour-Wright
(computer-vision) had been a generalisation of the target constraint value artificial-intelligence,
this explanation would be valid."
Status: incomplete
Rationale: use-alternative-resource
Strategy: generalization-constraint-values

Figure 10: Rationale-establishment-records for Exception-record-2

Unit-name: Constraint-value-disjunct-addition16
Unit-comment: "Modify a current-constraint by adding an additional-value
in a disjunct-clause to produce a new-constraint."
Amendment-summary: "Replace the current constraint
(area-of-expertise Seymour-Wright artificial-intelligence) with the new constraint
(or (area-of-expertise Seymour-Wright artificial-intelligence)
(area-of-expertise Seymour-Wright computer-vision))."
Implementation: (add-values-to-constraint (computer-vision)
(area-of-expertise Seymour-Wright artificial-intelligence))

Unit-name: Constraint-value-taxonomic-expansion17
Unit-comment: "Modify a current-constraint by replacing the existing-value with
its taxonomic-expansion to produce a new-constraint."
Amendment-summary: "Replace the current constraint
(area-of-expertise Seymour-Wright artificial-intelligence) with the new constraint
(area-of-expertise Seymour-Wright
(or artificial-intelligence distributed-ai planning robotics computer-vision))."
Implementation: (replace-values-in-constraint
(artificial-intelligence distributed-ai planning robotics computer-vision)
(area-of-expertise Seymour-Wright artificial-intelligence))

Unit-name: Target-object-attribute-value-addition18
Unit-comment: "Modify a KB object by adding a new-value to
the existing-values in an object-attribute field of the object."
Amendment-summary: "Add the new value artificial-intelligence to the current values
(computer-vision) of the area-of-expertise field of the KB object Seymour-Wright."
Implementation: (add-attribute-values-to-kb-object
(artificial-intelligence) (area-of-expertise Seymour-Wright))

Figure 11: Amendments for *Specialization-constraint-values-expl-14*

is based on the philosophy of the second approach, extending the database in order to fit the constraints. A *constraint partitioning* algorithm [Bro90] is applied to the unsatisfied constraints to determine a minimal amount of information that can be added to the knowledge base in order to remove the violation. The user selects one of these amendments to indicate how the explanation should be implemented. The knowledge base modifier performs the indicated changes specified in the implementation field of the amendment, and planning is resumed.

Note that since the specialization relationship (computer-vision is a subclass of artificial-intelligence) exists in the knowledge base, generalization-constraint-values-expl-15 and specialization-constraint-values-expl-14 could not possibly both be complete. An object cannot be both a generalization and a specialization of the same object. To illustrate the handling of incomplete explanations, suppose, however, that no taxonomic link exists between computer-vision and artificial-intelligence in the knowledge base. If that case, both explanations in Figure 10 would have been constructed as *incomplete*, but potentially valid, explanations. Either one of them may have been chosen during the negotiation process and completed by verifying with a knowledgeable agent that the appropriate missing taxonomic link could indeed be added to the knowledge base, resulting in an augmented static domain model.

7 Status and Conclusions

Plans generated by a nonlinear hierarchical planner in a realistic domain are based on an inherently incomplete domain model. Run-time discrepancies are therefore inevitable, and we have presented a constructive approach to handle such exceptions encountered during plan execution. With the empirical use of an interactive planning system, exception handling is regarded as a opportunistic entry point through which additional knowledge acquisition and refinement can take place.

Previous approaches have tended to view exceptions as destructive events, with no valid semantic basis. In contrast, we adopt the default assumption that the behavior of human agents is purposeful, even when it may be inconsistent with system expectations. A theory of *plausible rationales* has been operationalized for use as an exception detection and correction mechanism. Using this theory, we construct explanations of exceptional events and use techniques of plan network modification, constraint relaxation, and knowledge-base refinement to amend the ongoing plan, restoring consistency. Extensive replanning can be avoided, and new information can be acquired to refine the domain model.

A Common-Lisp prototype demonstrating our approach is implemented on a TI-Explorer. In addition, user studies were conducted which supported the adequacy and coverage of our extended exception taxonomy and the rationales used for explanation. Individuals experienced in conference planning were able to debug a sample task description, and the results indicated that most of the exceptions identified could be represented and identified by the techniques presented.

References

[AIS88] J.A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 83–88, 1988.

[BC87] C.A. Broverman and W.B. Croft. Exception handling during plan execution monitoring. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 190–195, 1987.

[BC88] C. A. Broverman and W. B. Croft. Plausible explanations to cope with unanticipated behavior in planning. Technical Report 88-56, COINS Department, University of Massachusetts, Amherst, Ma., June 1988.

[BL89] M. Beetz and L. Lefkowitz. Reasoning about justified events: A unified treatment of temporal projection, planning rationale and domain constraints. CSL Technical Report 89-6, Unversity of Massachusetts, 1989.

[Bro90] C. A. Broverman. *Constructive Interpretation of Human-Generated Exceptions During Plan Execution.* PhD thesis, University of Massachusetts, Amherst, MA, 1990.

[BW85] A. Borgida and K.E. Williamson. Accomodating exceptions in databases, and refining the schema by learning from them. In *Proceedings of the Very Large Data Base Conference 11*, pages 72–81, 1985.

[CL84] W. Bruce Croft and Lawrence S. Lefkowitz. Task support in an office system. *ACM Transactions on Office Information Systems*, July 1984.

[CLM84] Norman Carver, Victor Lesser, and Daniel McCue. Focusing in plan recognition. In *Proceedings of AAAI-84*, pages 42–48, 1984.

[Fik82] R.E. Fikes. A commitment-based framework for describing informal cooperative work. *Cognitive Science*, 6:331–347, 1982.

[Hay75] P.J. Hayes. A representation for robot plans. In *Proceedings IJCAI-75*, pages 181–188, 1975.

[Hol87] E. Hollnagel. Action not as planned: The phenotype and genotype of erroneous actions. Technical report, Computer Resources International, Copenhagen, Denmark, 1987. DRAFT.

[LC89] L.S. Lefkowitz and W.B. Croft. Planning and execution of tasks in cooperative work environments. In *Proceedings of the 5th IEEE Conference on Artificial Intelligence Applications*, 1989.

[Mah90] D. E. Mahling. *A Visual Language for Knowledge Acquisition, Display and Animation by Domain Experts and Novices.* PhD thesis, University of Massachusetts, February 1990.

[MSC89] D.E. Mahling, O. A. Sandvik, and W.B. Croft. Visual interaction between end users and goal based systems. CSL Technical Report CSL-90-4, CSL, COINS Department, University of Massachusetts, 1989.

[Nor81] Donald A. Norman. Categorization of action slips. *Psychological Review*, 88(1):1–15, January 1981.

[Ras80] J. Rasmussen. What can be learned from human error reports? In K. Duncan, M. Gruneberg, and D. Wallis, editors, *Changes in Working Life*. John Wiley, London, 1980.

[RDF85] S. Rajamoney, G. DeJong, and B. Faltings. Towards a model of conceptual knowledge acquisition through directed experimentation. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 688–690, Los Angeles, CA, 1985.

[RM82] J. Reason and K. Mycielska. *Absent-Minded? The Psychology of Mental Lapses and Everyday Errors.* Prentice-Hall, Inc., 1982.

[Sac77] E.D. Sacerdoti. *A Structure for Plans and Behavior.* Elsevier North-Holland, Inc., New York, NY, 1977.

[SD87] R. Simmons and R. Davis. Generate, test, and debug: Combining associational rules and causal models. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 1071–1078, Milan, Italy, August 1987.

[Sim88] R. Simmons. A theory of debugging plans and interpretations. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 94–99, 1988.

[SMR86] A. Sathi, T.E. Morton, and S.F. Roth. Callisto: An intelligent project management system. *AI Magazine*, 7(5):34–52, Winter 1986.

[Wil88] D.E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan-Kauffman Publishers, San Mateo, CA., 1988.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Handling Conflict Between Domain Descriptions With Computer-Supported Negotiation

STEVE EASTERBROOK
Department of Computing,
Imperial College of Science, Technology and Medicine,
180 Queens Gate, South Kensington, London, UK., SW7 2AZ.
<e-mail: sme@uk.ac.ic.doc>

Conflict is an inevitable part of both knowledge elicitation and system design. People will disagree over how to interpret features of the application domain, what the requirements for a new system are, and how to meet those requirements. Conventional systems analysis techniques avoid such conflicts, making any resolution untraceable and adding to the communication problems. This paper surveys a number of fields which have addressed the problems of conflict resolution. A model of computer-supported negotiation is presented which, with suitable tool support, can be used to fill the need to address conflicts in systems analysis directly. The model begins with an exploratory phase, in which the conflict is broken down into its components, eliciting the issues which underlie disagreements and criteria to measure their satisfaction. A set of options for possible resolutions are generated using design techniques. Finally, these options are compared to the original issues, and evaluated according to the criteria associated with the issues. The model is highly interactive, emphasising communication, and encouraging investigation of other viewpoints. Tools to support the knowledge acquisition, organisation and presentation aspects of the model are discussed.

1. INTRODUCTION

This paper discusses the occurrence of conflict in the software engineering process, and how it might be handled. In the computing literature, mention of the need to handle conflict is rare, which is perhaps surprising given the importance attached to it in the social sciences. For many years it has been recognised in management science and sociology that conflict is an inevitable feature of group interaction, to be harnessed for it's positive aspects, rather than suppressed [Robbins 1989], [Deutsch 1973], [Strauss 1978]. Some recent software engineering research has identified conflict as an issue [Curtis, Krasner & Iscoe 1988], [Anderson & Fickas 1989], [Feather 1989], [Robinson 1990], although as yet little progress has been made towards understanding how such conflict might be handled.

Examples of many different types of conflict abound in the literature on organisational behaviour. For example, Robbins [1974] describes a newly elected city manager who has promised an immediate improvement in rubbish collection. After several months the citizens complained that there was no improvement. On investigation it turned out that the citizens regarded "improved service" to mean more frequent collection, whereas the city manager had meant earlier, quieter and more economical collection. Thomas [1976] illustrates a different type of conflict, in an example of a manager who wanted his staff to adopt a new form for their weekly reports, whilst they preferred the old form. Again, investigation revealed the underlying cause: the staff didn't feel they had the time required to complete the new form, while the manger needed the extra information which wasn't on the old form. Both these examples are typical of organisational conflict, and it is not difficult to see how similar conflicts arise during the introduction of a new software system. Less obvious perhaps, is that the group developing the software is an organisation too, and will also be subject to such conflicts during the development process.

Typical Software Engineering methodologies do not explicitly address the resolution of conflict. Such methodologies are geared towards maintaining a consistent set of descriptions of a system, and the gradual transformation of these descriptions from the initial requirements stage through to

design and implementation. Indeed, we could characterise software methodologies as conflict avoidance, in that they prescribe particular approaches, which assist software practitioners in breaking problems down and resolving design decisions in particular ways. In this way the uncertainty of the designers is reduced by the provision of the collective wisdom embodied in the methodology [Lehman 1990].

While this approach helps prevent conflicts in directing the design process, it does not help to deal with conflicts in the requirements. Most methodologies, and especially those which rely heavily on formal methods, require a single consistent specification as a basis for a coherent design. In fact the trend towards formal methods underlines this avoidance of conflict. Formal languages are intended expressly to prevent ambiguity and inconsistency [Finkelstein, Maibaum & Finkelstein 1989]. While the ability to test specifications mathematically is a huge step towards detecting conflicts, it also carries the implication that conflicts are bad and should be eliminated. Methods developed to support formal specification reflect this philosophy, and miss the chance to explicitly address conflict resolution by exploiting the ability of formal representations to detect conflicts.

If much of software engineering is geared towards conflict avoidance, this in itself may not be a problem, for two reasons. Firstly, conflict may not be as extensive in software engineering as we have suggested, so that studying it might only help in exceptional cases. Secondly, avoidance is a valid way of tackling conflict, especially where it prevents energy being wasted on fruitless confrontation. In this case we need to show that the conflict is important enough not be be avoided, and that a project can be enhanced by handling conflict in more direct ways.

We argue in this paper that conflict is extensive in most real domains, and that conflict avoidance is not a satisfactory approach to producing specifications. Earlier, we suggested a distinction between conflict within the organisational setting into which the software will be introduced *(application domain conflicts)*, and conflicts within the organisation developing the software, which we term *software process conflicts*. In this paper we concern ourselves mostly with the former, leaving open the question of whether avoidance is appropriate to handle software process conflicts.

1.1. Sources of Conflict

It has been demonstrated that conflict is an inevitable feature of group behaviour [Robbins 1989]. We can therefore assume that any application domain involving more than one person will be subject to typical group conflicts. While it might be argued that a design process with a single goal, perceived in the same way by all participants, might be free of conflict, few real design processes are of this nature. This immediately suggests two possible sources of conflict in a real-world design process: conflict between the participants' perceptions of the domain, and conflict between the many goals of a design.

The extent of conflict in software engineering has recently been revealed by a major field study of software projects [Curtis, Krasner & Iscoe 1988]. Focussing on the behavioural aspects of software design, this study identified three major problem areas: the thin spread of application domain knowledge; fluctuating and conflicting requirements; and breakdowns in communication and co-ordination. We contend that each of these problems can be eased with tools that directly address conflict. A good conflict resolution approach necessarily emphasises communication between parties, which is fundamental to the solution of these problems.

Conflicting and fluctuating requirements have many causes, from change in the organisational setting and business milieu, to the fact that the software will be used by different people with different goals and different needs. Handling constant change in requirements (which has been termed *requirements maintenance* [Finkelstein *et. al.* 1989]) requires an evolutionary approach that must be based on accurate capture of rationales and process information.

Unless the application domain with which the software deals is free of conflict, then the resulting software must incorporate this conflict. For small programs, the domain can be restricted until the

conflict is excluded. For any large scale software, this is not practical. When the application knowledge is spread over many people, there is likely to be much disagreement between them, and fitting together the many contributions will inevitably lead to conflict.

Even if a domain appears to be free of conflict, quite often there will be areas in which there are different ways of looking at things. While such perspectives may not be fundamentally incompatible, they are likely to appear inconsistent, and so lead to conflict. Other sources of conflict include: conflicts between suggested solution components; conflicts between stated constraints; conflicts between perceived needs; conflicts in resource usage; discrepancies between evaluations of priority, merit, etc.

1.2. Conflict Resolution

There are a number of problems with any method which avoids conflict. As there is no means of expressing conflict within the method, where it does exist it is suppressed. The inevitability of conflict makes it's suppression all the more problematic: the resolution must be carried out outside the framework of the method and consequently is likely to be carried out at an inappropriate time, using an undesirable means. In addition, resolution thus achieved is untraceable, making rationales invalid, and results irreproducible.

Failure to recognise conflict between the perspectives of the participants will lead to confusion during the analysis phase, which will continue throughout the lifecycle. Often, a single perspective will be adopted as the basis for specification at the cost of any alternative perspectives. In this process useful ideas associated with the rejected perspectives will be discarded, along with the goodwill of their originators.

These problems make a study of conflict resolution desirable. We can draw on many fields which have addressed some or other aspect of the problem, in order to understand the processes involved. The next section surveys research on conflict in relevant fields, discussing in each case how the work might apply to software engineering. The third section synthesizes a model of conflict resolution, using negotiation, which can be used as a basis for computer support.

2. SURVEY OF RELEVANT FIELDS

This section reviews a number of fields in which conflict and its resolution are studied. These fields fall roughly into two traditions, the mathematical and the behavioural, which approach conflict in very different ways. We also survey work related to conflict resolution within fields of computer science and software engineering.

2.1. Terminology

Before we survey other fields, it is useful to introduce a suitable terminology. We will talk about conflict between *parties*, as conflict may occur between individuals, groups, organisations, or even between different roles played by one person. Similarly, when discussing conflict resolution, we will refer to *participants* of the process, to cover a similar diversity. Not all parties to a conflict need necessarily be participants in its resolution.

The term *conflict* is used in its broadest sense, to cover any interference in one party's activities, needs or goals, caused by the activities of another party. The situation can be characterised as a disparity between the goals of the parties, although this would not be considered conflict until the effects are felt. This use is at odds with the more restricted use in political science as the opposite to co-operation as a mode of interaction, but corresponds more closely to its use in organisational behaviour. Conflicts do not necessarily need to be resolved, but where they are, the possible resolution methods range from the co-operative to the non-cooperative.

The approach used to settle a conflict is a *Resolution Method*. Methods include negotiation, competition, arbitration, coercion, and education [Strauss 1978]. Not all conflicts need a resolution

method, as not all conflicts need to be resolved. Three broad types of resolution method can be distinguished: *Co-operative* (or collaborative) methods, which include negotiation and education; *Competitive* methods, which include combat, coercion and competition; and *Third Party* methods, which include arbitration and appeals to authority.

Negotiation is a collaborative approach to resolving conflict by exploration of the range of possibilities. It is characterised by the participants attempting to find a settlement which satisfies all parties as much as possible. Such an approach has been variously termed *integrative behaviour* or *constructive negotiation* (to distinguish it from *distributive*, or *competitive* negotiation). This definition of negotiation is not universal. Authors such as De Bono [1985] restrict negotiation to its distributive variety, implying a process of bidding and concession-making, and so attack it as being inferior to an integrative approach. We prefer to give negotiation its broader definition, and call the concession-making process *Bargaining*.

In contrast to negotiation, *Competition* concentrates on achieving maximum satisfaction for a participant, without regard for the degree of satisfaction of other parties. However, a competitive approach is not necessarily hostile. An extreme form of competition is when all gains by one party are at the expense of others, which, in Game Theory, is termed a zero-sum game.

Third Party Resolution covers any situation where participants are unable to resolve a conflict between themselves, and so have to appeal to an outside source, whether this be the rule-book, a figure of authority, or the toss of a coin. Such a situation can occur with the breakdown of either negotiation or competition as resolution methods. There are two types of third party resolution: Those in which the cases presented by each participant are taken into account, which we might term *judicial*; and those where a decision is determined arbitrarily (E.g. tossing a coin), or by factors other than the cases presented (E.g. by the relative status of the participants), which we might term *extra-judicial*.

Bidding and *Bargaining* are phases of the resolution process. Bidding is where participants state their desired terms for the settlement, often with an indication of the relative importance of them, as a basis for bargaining. Bidding takes place in some form or other in most resolution methods, as participants must present their side, although in methods such as coercion, the bidding might be one-sided and implicit. A *position* is the set of terms that a participant commits itself to by making a bid. In bargaining, participants search for a satisfactory integration of bids. In the simplest case this involves a converging sequence of bid and counter-bid, while at the other extreme, participants seek to blend complex bids together. Note that the description of the outcome as *satisfactory* depends on your viewpoint. However, bargaining usually results in a compromise, whereas true negotiation seeks to develop a new solution which fully satisfies all participants.

2.2. Mathematical and Economics Models

Decision theory is a prescriptive approach to analysis of a set of pre-specified alternatives. The interesting problems are concerned with resolving multiple conflicting objectives [Keeney & Raiffa 1976]. Decision theory assumes a single entity is making a choice, in contrast to conflict where there is more than one entity, each with a different perspective. Decision theory has a role in conflict resolution in helping participants to evaluate bids, to justify such evaluations, and to persuade the other participant(s) that a solution is satisfactory.

Bargaining theory is an attempt to produce descriptive models of bargaining processes, and is especially concerned with commerce and politics. Patchen [1970] surveys models of bargaining theory and notes that the more complete models include wider concerns than bids and outcomes, including how participants influence each other's behaviour, and factors such as the cost of various actions and the cost of delaying agreement. Bargaining Theory frequently makes use of the *joint outcome space* [Thomas 1976] as a tool for illustrating how the parties perceive the options in a conflict (Figure 1). Note that there may be possibilities not perceived by the participants, which provide better resolutions. Bargaining theory does not indicate how these might be found,

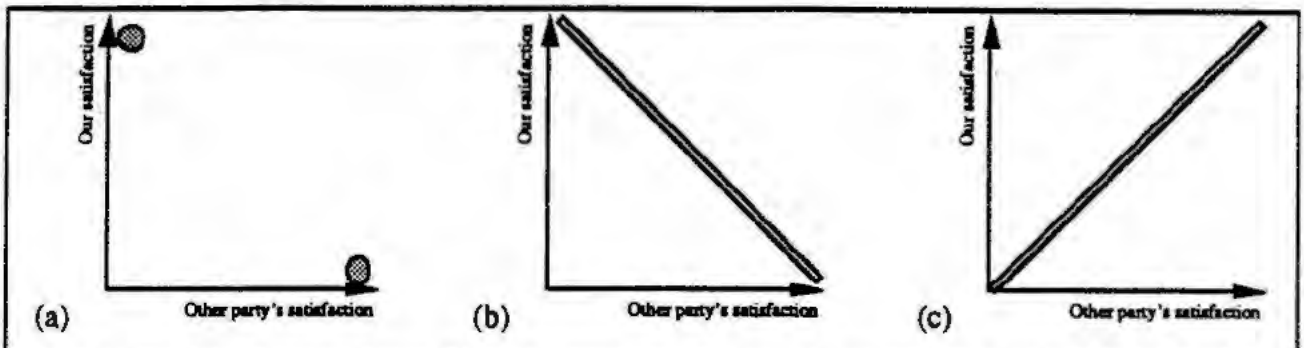


Figure 1: The Joint Outcome Space. These diagrams illustrate how the options are perceived in (a) a win/lose conflict, (b) a zero-sum conflict, and (c) a common problem. In (a) only two outcomes are perceived, one party wins and the other loses. In (b) there are a range of possible outcomes, but any gain by one party is totally at the expense of the other. By contrast, in (c), any gain by one party is also a gain by the other.

concentrating instead on the process of bidding and counter-bidding.

Game Theory is defined as the theory of rational decision in conflict situations [Rapoport 1974]. Participants are regarded as players, and game theory examines the strategies used by the players in the process of trying to achieve particular outcomes. In contrast to the joint outcome space, game theory often makes use of the payoff matrix (Figure 2). This reflects the assumption that the set of outcomes is known (though not necessarily finite), and that associated with each outcome is a calculable payoff for each player. Limitations of game theory include the restricted sets of available actions, and the assumption that the payoffs for any action are known with certainty by all players. However, game theory does produce some useful information about the kinds of strategy that can be used to induce co-operation and how various strategies pay off for the players [Axelrod 1984].

Group decision making is the normative study of how individual preferences can be combined into a group decision. Luce and Raiffa [1957] defined the problem as that of finding a method, or *welfare function* for combining individual preference rankings into a social preference, which satisfies properties such as fairness and representativeness. Work on Group Decision Making extends decision theory to cope with more than one decision maker, but still suffers from the assumption that all the options are known. In design, we are usually concerned with creating new options, rather than selecting among existing ones.

2.3. Behavioural Models

Conflict Theory studies the conflicting pressures of various groups in society, recognising that there are many different groups in society with different goals, and that conflict is a frequent occurrence. Strauss [1978] points out that the majority of conflicts are resolved by co-operative means, often unconsciously, and yet despite this, very little attention is paid to these co-operative mechanisms. Deutsch [1973] begins to examine the nature of conflict, and gives a list of issues involved in conflicts:

		Prisoner B	
		Not Confess	*Confess*
Prisoner A	*Not Confess*	1 year each	10 years for A and 3 months for B
	Confess	3 months for A and 10 years for B	8 years each

Figure 2: The payoff matrix for the prisoner's dilemma. Each player must decide, in isolation from the other, whether to confess to a crime that the judge is sure they both committed. By confessing each will implicate the other, and their joint best strategy is for both to keep quite.

– Control over resources
– Preferences and nuisances, where the tastes or activities of one party impinge upon another.
– Values ("what should be"), where there is a claim that a value, or set of values should dominate
– Beliefs ("what is"), when there is a dispute over facts, information, reality etc.
– The nature of the relationship between the two parties

This list closely follows those suggested by Robbins [1989], who adds that communication problems are a major cause of what he terms *pseudo-conflicts*, and De Bono [1985], who notes that the usual thinking styles encourage people to be contrary.

One of the major concerns of *Organisational Behaviour* is team-work within organisations, and how communication and co-ordination of teams can be effected. While early work tended to assume all conflict was undesirable, empirical work in the last few decades has demonstrated that conflict is an inevitable feature of group interaction. Robbins [1974], among others, has advocated that conflict management include not just resolution of conflict, but stimulation of conflict too. This is a result of observations that conflict has a useful role in organisations, in providing a stimulus to innovation, as it involves questioning and evaluating received wisdom. It is also a major weapon against stagnation and resistance to change.

In his model of the stages in the development of conflict in group behaviour, Robbins [1989] groups the conditions which allow conflict to arise as follows:
– Communicational, including: insufficient exchange of information; noise; and the semantic differences that arise from selective perception and difference of background.
– Structural, including the goal compatibility of members of the group; jurisdictional clarity; and leadership style.
– Personal factors, including: individual value systems; and personality characteristics.

Studies in organisation behaviour have shown that people are predisposed to tackle conflict in certain ways, according to their character rather than the context of the conflict. It is useful to identify these modes, and examine the utility of each. Thomas [1976] describes five orientations towards conflict handling (Figure 3), based on areas of the joint outcome space:

Competitive - one participant seeks to dominate the process, without regard for the others. It is useful for quick decisive action, or where unpopular actions are needed for important issues.

Collaborative - participants seek to understand their differences and achieve a mutually beneficial solution. It is appropriate where participants' insights and commitment are important and need to be merged rather than compromised.

Avoidant - the conflict is recognised to exist but is suppressed by one or more parties, or handled by withdrawal. It is useful where an issue is unimportant, where the potential disruption would outweigh the benefits of resolution, or where information gathering is more important.

Accommodative - a party becomes self-sacrificing to appease another, and places the other's interests above their own. It is useful when issues are far more important to one party than another, where one party is losing and needs to minimise loss, or simply to build harmony and gain social credits.

Sharing - each party makes some concessions in order to reach a compromise. It is appropriate where temporary settlements or expedient

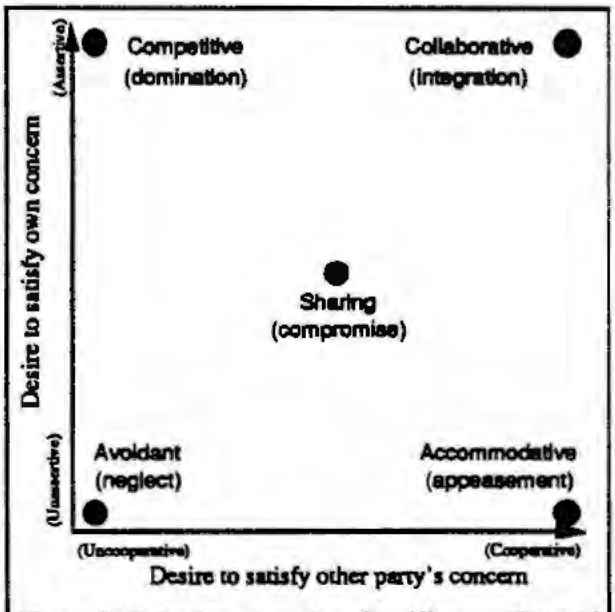


Figure 3: Behavioural modes of tackling conflict with (in brackets) the outcome sought in each mode.

solutions are needed especially under time pressure, or where goals are directly opposed. Each mode is appropriate in some circumstances, and it is useful to compare these modes with methods of resolution. For example, education, while normally a collaborative method, can be used to achieve avoidance or accommodation by enabling participants to understand their differences better. Negotiation can assist with achieving a compromise, seeking an accommodation, or regulating competition. It is likely that successful negotiation requires at least some assertiveness and at least some co-operation from each participant, implying each participant must have some motivation to resolve the conflict rather than avoid it.

A number of models for conducting face-to-face negotiation have been proposed (E.g. Scott [1988], Fisher & Ury [1981], De Bono [1985]). Scott [1988] gives advice for preparation and the opening moments (setting the climate and procedure) of a negotiation. He uses a four stage model to pace the negotiation: Exploration; Bidding; Bargaining; and Settling. The exploration stage is emphasised as the most crucial, allowing participants to explore a range of possibilities before any confrontation takes place. In particular, it allows the participants to explain to each other their interests (Figure 4), and discover shared goals which were previously obscured from both.

De Bono [1985] discusses the flaws in argumentation that render it ineffective as a means of negotiation. The adoption of a particular perspective (or theory) dictates how the world will be perceived and leads to a rejection of alternative theories, making argumentation a polarising process. The key to De Bono's method is the use of a third party to *design* solutions to conflict, as opposed to fighting, negotiating or problem solving. However, his complaints against negotiating and problem solving are based on very narrow definitions of these methods. Stefik *et. al.* [1987] suggest that removing the personal attachment to positions prevents polarisation. Their computerised meeting room allowed participants to dispense with the feeling of ownership of ideas, and so reduce the associated emotions when ideas are discarded or adopted. Similarly, Fisher & Ury [1981] recommend that rather than bargaining over positions, participants should focus on interests, and investigate options for mutual gain.

Some of the recommendations made in these negotiation models are clearly useful for design. To summarise the main points:

- Exploration of each other's perspective is essential to constructive negotiation
- Participants should be separated from the bids in order to avoid polarization.
- The negotiation must begin by examining interests rather than positions so as to achieve a better understanding of the conflict.
- There is a need to generate many options without discarding any prematurely.
- The procedure (and criteria) for selecting a good solution must be agreed on.

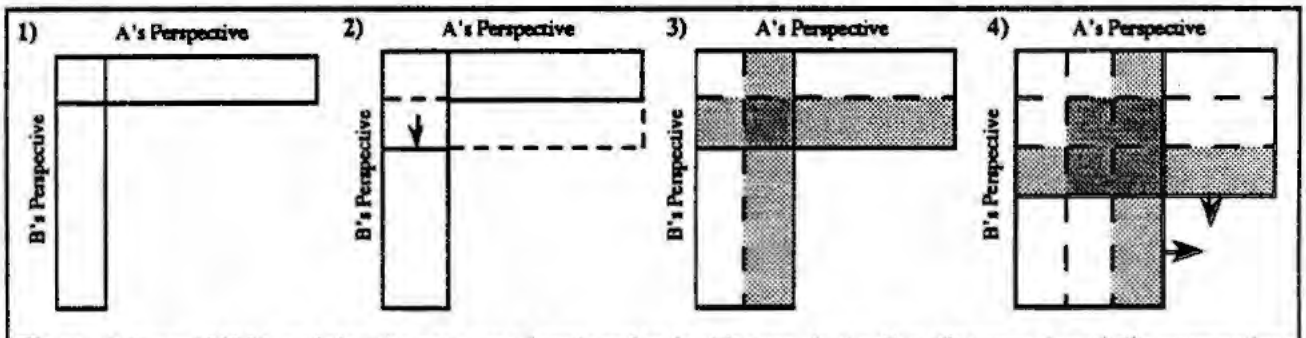


Figure 4: Scott [1988] explains the process of exploration in this way. In the first diagram, A and B's perspectives are very different, with little overlap. In (2), B begins to explain her perspective to A. Because A's perspective is different, A is able to take a great leap forward. In the process of learning more about the each others' perspectives, they discover joint interests, shown by the darker shaded areas. These areas are likely to yield a jointly acceptable solution to the conflict.

2.4. Conflict in Computing Science

Various field of computer science have addressed the problems of conflict. The ubiquity of conflict in the real world means that attempts to model the world, most notably in AI, have to deal with it. Also, systems designed to support human interaction must take account of the conflict between people.

Knowledge-based systems rely on consistent knowledge for their inference mechanisms to work. In many cases, this is achieved by only consulting a single expert. Inconsistencies are attributed to mistakes in the acquisition process and are eliminated through a lengthy process of interactive debugging and refinement [Davis 1979]. This insistence that expertise must be consistent and rational imposes restrictions on the knowledge acquired. The knowledge acquisition process becomes not so much the modelling of the expert's behaviour, but the synthesis of a domain model which need not resemble any mental model used by the expert [Shaw & Woodward 1989]. In this way conflicts in the expert's knowledge can be avoided. When further experts are involved, the problems of conflict cannot be avoided so conveniently. There is little pressure on groups of experts to agree with one another, and the synthesis of a consistent domain model can be very difficult.

Shaw & Gaines [1988] compare the Entity-Attribute models of different experts, and identify four types of comparisons between conceptual systems:

Consensus - experts use the same terminology to describe the same concepts
Correspondence - experts use different terminology to describe the same concepts
Conflict - experts use the same terminology to describe different concepts
Contrast - experts use different terminology to describe different concepts

Each of these situations can be useful in capturing different perspectives, and in particular, the availability of alternative terminologies makes a knowledge-base more accessible. Whilst Shaw's use of the term conflict is rather different from ours, it does highlight the way that terminological differences can obscure the deeper agreements and conflicts between people. Shaw describes a methodology for recognising each of the four situations using entity-attribute grids. However, this requires the participants to agree first on the definition of a common set of entities, and is concerned with differences in the way experts distinguish between the entities. It is not clear how this methodology might be extended to the types of representation used in systems analysis, where the establishment of an agreed set of entities is itself a problem of conflict resolution.

Distributed Artificial Intelligence (DAI) questions the usual AI assumption that a single self-consistent entity (such as a conventional knowledge base) can demonstrate intelligence. Problem-solving activities can be divided up among agents, according to their specialist knowledge, with the premise is that intelligence is an emergent feature of co-operative behaviour [Huhns 1987]. The paradigm has a natural ability to handle conflicting knowledge without the usual logical contortions arising from inconsistency, by allowing agents to develop and maintain several alternative hypotheses at once. This is demonstrated in the blackboard system [Nii 1986], in which separate knowledge sources communicate partial hypotheses using a shared *blackboard*. The knowledge sources modify and extend existing hypotheses on the blackboard.

Most DAI systems assume benevolent agents working towards the same goal. Rosenschein [1985] notes that in real world situations, perfect co-operation never happens, as the goals of any two agents will never coincide exactly. He uses payoff matrices from game theory to compare goals, and discusses various situations in which conflict of goals can occur, and how they can be resolved [Rosenschein and Genesereth 1985].

DAI has not progressed much beyond game theoretical studies of interaction. Models which require resource conflicts to be resolved involve little more than bidding and bargaining strategies. Keeping the contributions separate and applying them as separate reasoning systems avoids conflict, reducing it to a problem deciding which rule should be selected when several are applicable, where the possible combinations are foreseeable. Models which allow multiple

hypotheses to be developed accept the first hypotheses to meet certain criteria, where the criteria used are built into the system. Work is needed to study how to combine elements of conflicting hypotheses as the basis for better hypotheses. In order to achieve this, the rationales behind each element of the hypotheses are needed.

Computer-Supported Co-operative Work (CSCW) studies how computers can be used in collaborative activities. Greenberg [1989] divides CSCW into two areas: Real-time collaboration and asynchronous collaboration. The former studies how computers are used to provide electronic media for use in face-to-face meetings, and to facilitate remote conferencing. Asynchronous systems co-ordinate group working over longer periods, and provide new communication channels between colleagues. These include electronic mail, bulletin boards and hypertext.

CSCW provides a number of tools intended to improve group collaboration and hence manage conflict. Xerox PARC's *CoLab* [Stefik et al 1987] is a testbed meeting room which uses high resolution screens to replace the role of the whiteboard. Two of the software tools provided are of interest here: *Cognoter* and *Argnoter*. Cognoter is used to develop outlines for talks and papers collaboratively, and divides meetings into three phases: *brainstorming*, *organising*, and *evaluation*. Argnoter is intended for use in evolving designs, and attempts to overcome three major causes of dispute: personal attachment to positions, unstated assumptions, and unstated criteria. Proposals are presented using webs of interconnected windows, often modifying existing proposals, or combining features of several. Reasons for and against are then linked to each proposal. Finally, the assumptions made by the arguments and criteria for decision making are elicited. The assumptions can be grouped together into *belief sets*, to characterise points of view, and to explore the consequences of those views. Stefik *et. al.* [1987] compare the tool to a spreadsheet, in that it doesn't understand the proposals and arguments that it manipulates, but can compute the logical relationships between them.

Hypertext offers the ability to support the collaborative elicitation and organisation of ideas over longer durations [Conklin 1987]. As Schuler [1988] points out, hypertext provides an excellent vehicle for supporting (but not supplanting) negotiation. Differing opinions and viewpoints can be represented and linked in the same system, encouraging plurality rather than stifling it. Lowe has built a hypertext system, SYNVIEW, based on a model for debate and reasoning [Lowe 1985]. By concentrating on debate, the system can contain all viewpoints rather than just a dominant one, with any group decision-making or voting based on access to a common body of material. Conklin [1989] takes this even further, and models positions, issues and arguments explicitly.

2.5. Representation of conflict in Software Engineering

A number of recent studies have concluded that conflict has an important role to play in software engineering. For example, Curtis, Krasner & Iscoe [1988] identify *exceptional designers* who have a deep understanding of both the application domain and the design process which enables them "to integrate different, sometimes competing perspectives on the development process". If the interaction of perspectives is to be supported, they need to be explicitly modelled. Each participant brings a number of preconceptions or biases into the software specification process, together with a certain amount of knowledge. The areas of knowledge do not fit together like a jigsaw, but instead overlap in some places, conflict in others, and often leave gaps. The process of elicitation and specification of requirements can be regarded as a knowledge acquisition and knowledge management task [Easterbrook 1989].

Finkelstein *et. al.* [1989] formalise the notion of a perspective into what they term a *ViewPoint*. Each viewpoint represents some area of knowledge and a preferred representation for that knowledge. More specifically, a viewpoint has the following components:

- A style, which is the representation scheme used,
- An area of concern, or domain,
- A specification, which is the set of statements in the viewpoint's style describing the area of concern,

– A work plan, which describes how the specification can be changed.

This model abstracts away from the people involved, allowing one person to have several viewpoints (as a person may have several areas of concern), and also for one viewpoint to represent several people (where people share an area of concern).

Finkelstein & Fuks [1989] use the viewpoint as a basis for a study of some of the communicational problems surrounding conflict. They describe a formal model of dialogue between two agents, which allows agents to share knowledge and detect inconsistencies between their knowledge. The dialogue consists of speech acts, such as statement, question, challenge, or withdrawal, and rules constrain which acts are legal in which contexts. Agents can query chains of reasoning made by other agents, and request information which they need to verify the conclusions. In this way, conflicts based on misunderstandings and incomplete knowledge can be detected and resolved.

Feather [1989] uses a basic specification as a point of departure for development along separate lines of concern. At some later stage these are merged to produce a single specification, which will then reflect all the concerns. This model has the benefit of delaying the resolution of conflict between separate concerns until after the information gathering stage. While it is not yet entirely clear how best to merge the parallel elaborations, Feather has examined the different types of conflict that occur.

One approach to easing the integration of separate specification components is through tools which support negotiation. Robinson [1990] describes tools that allow a single arbitrator to evaluate the preferences expressed by various perspectives, and guide the search for new solutions which satisfy all perspectives. A single domain model is used, expressed as a hierarchy of goals in which perspectives associate different values with the goals. Integration involves searching for novel combinations of proposals, which increase the satisfaction of all perspectives' goals. This is done using a joint outcome space on which an ideal, but probably unachievable combination of perspectives is used to stimulate consideration of other combinations that come close to this ideal.

These approaches to software specification all question the assumptions made by conventional software models which ignore conflict. Clearly, more work is needed to clarify how conflicts based on differing requirements can be resolved. One major issue is the need to establish common ground between viewpoints. No conflict resolution can occur until participants have enough common ground to communicate. In many of the above models, the common ground is assumed.

A second issue is how the resolution is devised. If all possible conflicts are foreseeable, then the resolutions can be enumerated beforehand, either directly, or using a set of heuristics. In general the design process is not predictable, and sociological studies of negotiation indicates that good resolutions require creative input. This poses problems for automatic tools. Anderson & Fickas [1989] suggest that in well charted domains, the experts will be aware of the conflicts and how to deal with them, which suggests that resolutions can simply be elicited from experts. However, standard solutions are not necessarily the best, nor will they always work in the environment introduced by computerisation. The wealth of research on conflict in organisational behaviour suggests that at least in this field, resolutions are not well known.

2.6. Summary

We have discussed three broad areas in which conflict and conflict resolution are studied. The first of these, which we have labelled *mathematical models*, includes areas of decision theory and game theory. These fields are highly theoretical, and make some restrictive assumptions about, for example, the state of knowledge of the participants, and their motivations. Nevertheless, research in decision theory has developed a number of tools which can be used to clarify the issues in decision making. Game theory and bargaining have laid the groundwork to evaluating and understanding the use of strategies in conflict situations.

Behavioural models, from the social sciences, show more promise in channelling conflict

situations into creative processes, but give little indication how such processes might be supported with computational tools. The main results from research in these areas is that not only is conflict inevitable in society, both within and between individuals and organisations, but that conflict has a useful role in facilitating change and producing higher quality group decisions.

Thirdly, we have covered areas within computer science which have studied conflict. Most of these fields are extremely young, some less than a decade old, and are only just recognising the role of conflict management. The fields we have covered seem to fall neatly into two camps: the AI approach, which attempts to automate completely the resolution of conflict between agents, based on mathematical approaches; and the behavioural approach, in which computers are used as tools for supporting human conflict resolution, usually in a collaborative manner.

Finally, we have surveyed recent work in software engineering that recognises the existence of alternative and possibly conflicting views in requirements analysis. These models capture conflict by allowing different views to be represented, removing the conventional assumption that conflict can be avoided in software engineering. However, as yet no clear model exists of how conflict resolution might be modelled using these frameworks.

There is a consensus of opinion in work on group behaviour that good solutions to conflict require creative input. If this is the case, and it certainly seems likely in design contexts, then it is unlikely that algorithmic conflict resolution methods can be provided. Rather, we can provide tools for analysing conflict situations and for encouraging co-operative and creative approaches to conflict resolution. The formal techniques developed in mathematical models of conflict can be used in these tools to supplement the creativity encouraged in behavioural models.

3. A MODEL FOR CONFLICT RESOLUTION

We now present a model for the management of conflict in the software process. The twin goals of encouraging conflict and providing productive resolution methods form the basis for this model. In this paper we concentrate mainly on the latter goal, noting that the provision of explicit resolution methods and the associated communication channels should in itself begin to encourage the expression of conflict. We are currently building a set of tools to support this model, the nature of which we will briefly describe.

The model is intended to be prescriptive in that it acts as a set of guidelines, without being a rigid formal process. The tools used to support the model are highly interactive, and are designed to provoke discussion of the conflict situations as much as elicit a suitable resolution. The model is based on the behavioural approaches used in organisational design, and in particular takes note of the need to separate the people from the problem, in order to avoid the polarising nature of arguing from entrenched positions. There are three phases in the model: an Exploration phase, a Generative phase, and an Evaluation phase. We discuss each of these phases in more detail in the rest of this section after a discussion of the context for this model.

3.1. The Occurrence of Conflict in Software Design

As we noted in the introduction, avoidance seems to be the main reaction to conflict in existing software process models. There are several reasons why this ought not be the case. We have discussed the inevitability of conflict, and how it's suppression leads to untraceable resolution. Research into group behaviour indicates that conflict can produce higher quality solutions. This is a strong argument for conflict to be nurtured in the software process, and participants need to be encouraged to express divergent views. This will ensure that the resulting system does not reflect just one point of view, and does not ignore those concerns which interfere with the dominant concern.

Clearly, in software design, we want to promote collaborative work. It is vital that there be no losers from any conflict in the specification process, as the commitment of all participants must be maintained. Hence, encouragement of conflict must be matched with resolution methods which

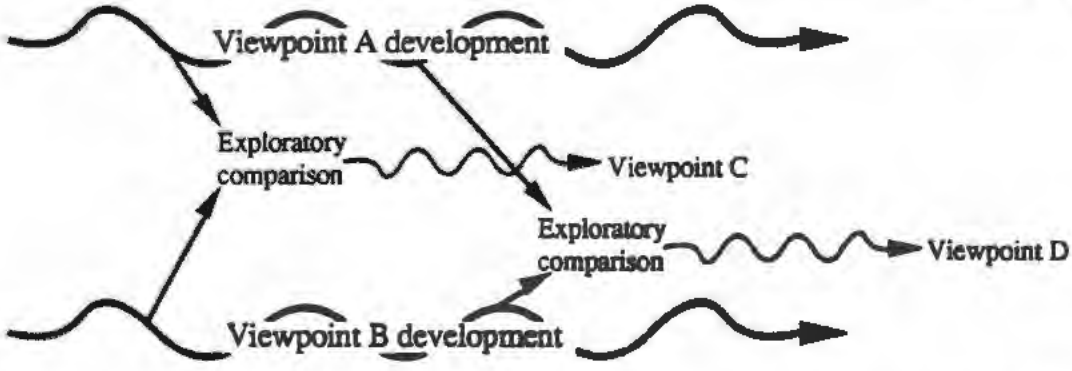


Figure 5: Our model of parallel viewpoints allows exploratory comparisons of viewpoints as they are elicited. These then may be discarded, or developed as new viewpoints.

strive to satisfy all parties. An integrative approach should be encouraged, to ensure that when divergent views arise they are incorporated into the process. The ultimate goal of a project should be to produce a result which maximises the satisfaction of all concerns.

The two key collaborative methods for conflict resolution are (integrative) negotiation and education. Both rely heavily on communication, and both greatly ease conflicts based on communication problems. As Robbins [1974] notes, increased communication leads to decreased conflict up to a certain level, but that too much communication can lead to increased conflict. The explanation is that a certain amount of communication allows participants to discover commonalities and iron out perceived conflicts. However, a high level of communication highlights the details on which participants disagree. Such conflicts are likely to be well-founded, and productive, and should not be discouraged. We should therefore strive to convey as much knowledge as possible to the participants by educating them about each other's points of view.

3.1.1. The Specification Context

We use a notion similar to the viewpoints proposed by Finkelstein *et. al.* [1989] as a basis for our model of the specification process. Because viewpoints do not correspond to people, they take account of the conflicts between a single person's roles, as well as inter-person conflicts. The term *viewpoint* refers to a formal object and we use the term *perspective* informally, to refer to the contribution made by a viewpoint. Each viewpoint has an *originator* – the person(s) associated with the viewpoint, and contains a self-consistent description of an area of knowledge.

The elicitation of a viewpoint does not occur in isolation from other viewpoints, as comparisons can shed new light on a particular perspective. Hence at any point in the process, parts of two or more viewpoints might be compared. The comparisons may involve conflict resolution, but are intended to be exploratory – any resolutions generated need not be incorporated in the final specification. Feedback from these explorations can be used in the development of the viewpoints, for instance to modify terminology, or to elicit information that the originator neglected. The results of any exploratory integrations can be treated as new viewpoints which can continue to take part in the development process (Figure 5). Such derived viewpoints effectively represent coalitions of perspectives, which have been shown to arise in software projects [Curtis, Krasner & Iscoe 1988].

The modelling of viewpoints allows differences between perspectives to be captured and accommodated. If only a single description was maintained differences between parties would tend to be avoided or suppressed, and often go unnoticed. As the viewpoints contain formal descriptions, it is possible to combine parts of different viewpoints to reason with, and detect inconsistencies. The process of parallel development of viewpoints – with exploratory integrations being initiated at any point – provides the context for our model of conflict resolution.

3.1.2. Detection of conflict

The first problem for conflict resolution is to recognise that a conflict exists. This might be harder than it seems for a number of reasons. The terminology used by the participants is unlikely to match exactly [Shaw & Gaines 1988], and the style in which knowledge about an issue is expressed will differ. This difference may be because of different representation schemes, or different descriptions within the same representation scheme. Also, participants will have different areas and different amounts of knowledge, making it difficult to make comparisons. These problems make it hard to tell where participants are agreeing, let alone where they are disagreeing.

Our definition of conflict was based on interference: two parties are in conflict if the activities of one interfere with those of another. Hence, viewpoints are free to differ, and only conflict when that difference matters for some reason, leading to interference. There are a number of situations in which the differences matter:

- When viewpoints need to be compared.
- When there is a need to reason with knowledge from several viewpoints.
- When the originators insist their viewpoints are better than others (and so perhaps should be adopted at the expense of them).
- When a coherent description is needed for further progress.

Under normal circumstances, differences between viewpoints are ignored, allowing them to develop independently. By only entering the conflict resolution process when differences between viewpoints matter, we avoid attempting to resolve conflicts unnecessarily. A conflict, then, is simply a difference that matters.

There is one remaining problem with which this approach does not help. Although the conflict resolution process is only invoked when differences become important, this does not ensure that all relevant viewpoints participate in the conflict resolution. When a difference between two viewpoints becomes important enough to attempt to resolve, there may be other viewpoints which contain extra information relevant for the resolution. It is not clear how these relevant viewpoints can be detected, especially in the presence of the mismatches in terminology and style discussed above. At present, the model assumes that relevant viewpoints are included, and concentrates on the resolution.

3.1.3 An Example

Take for example the specification of a library system. An analyst is trying to establish a state transition diagram for the books of the library, and is offered two different versions by two different librarians (Figure 6). One gives a description based roughly on a book's physical whereabouts, whereas the other gives a description based on how a book can be accessed. There is a conflict between their views of the library.

While it may appear that there are a number of correspondences between the two descriptions,

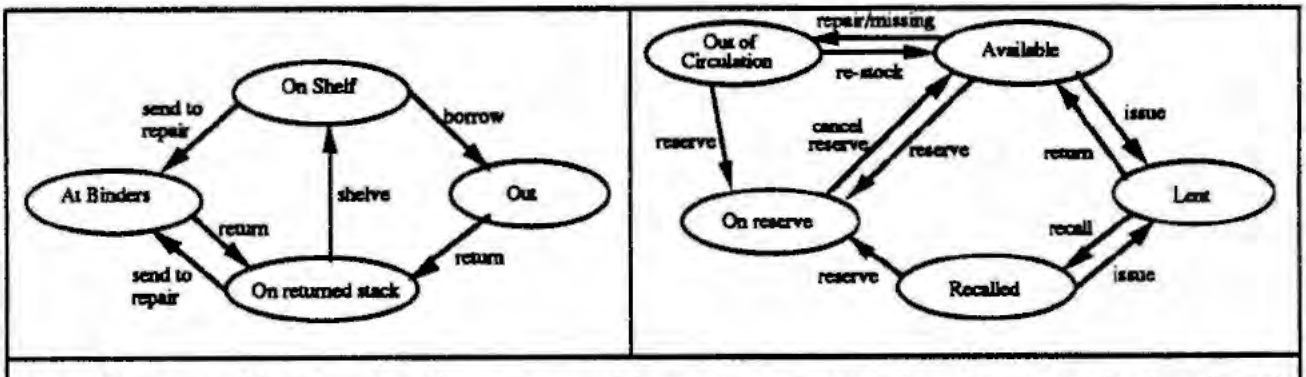


Figure 6: The possible states for a library book: (a) based on its physical whereabouts; and (b): based on the accessibility of a book.

these are not as simple as they seem. For example, the concepts ON SHELF and AVAILABLE are similar, except that the latter includes books waiting to be shelved: it assumes that librarians be able to locate unshelved books for loan. LENT and OUT are also similar, except that the former includes books being used within the library, while the latter only includes such books if they are from the reserve collection. To make things worse, both could have used the same terminology.

3.2. The Exploration phase

The first phase of the model is exploration. The aim of this phase is to arrive at a better picture of the conflict. Essentially, this is a process of knowledge elicitation, as additional knowledge is needed about the descriptions in conflict. This involves identifying why the conflict occurred, and hence the type of conflict, the extent of the conflict, and the issues involved. Such information can be represented in a manner of ways ranging from formal to informal, with hypertextual links illustrating connections between items both within perspectives and between perspectives. In particular, links illustrating correspondences and conflicts can be used.

The exploration phase begins once a conflict has been detected. The information available consists of the relevant parts of the viewpoint descriptions, and an indication of where the conflict was detected. For example, given the descriptions in figure 6, let us say the analyst is trying to establish when a book is available for loan. The states ON SHELF in figure 6(a), and AVAILABLE in figure 6(b) seem to correspond roughly, but there is conflict, as neither the names, nor the transitions attached to these states match. In this case we begin the exploration with these two diagrams and an indication that the conflict is between ON SHELF and AVAILABLE.

3.2.1. Establishing Correspondences

The first problem is to establish some common ground between the descriptions. This is important to delimit the extent of the conflict, and to provide the participants with a basis for communication. Given two statements which are known to conflict, the context in which the statements were made can be compared to find out the extent of the conflict. Initially, this context can be provided by considering the statements which are directly related to the ones in question. In a graphical notation, these are the arcs and nodes connected to the areas in question, and for a chain of inference, the immediate antecedents and consequences can be used.

The problem is to identify correspondences between the items in the descriptions. Such correspondences may be exact, in that items are agreed as having the same definition, or approximate, where meanings are similar, but differ in certain details. Many terminological differences will be discovered at this point: methodologies for recognising terminological mismatches, such as that of Shaw [Shaw & Gaines 1988], as well as tools for detecting graph isomorphism, could be usefully employed here. Our tool support is less sophisticated, in that it simply presents the relevant descriptions (graphically, where possible) side by side, and allows users to create typed links between components of each description. The tool can prompt with suggestions of items which might be related, based on similarities in interconnectivity.

To illustrate this process, consider the library books example. The arcs attached to the bubbles ON SHELF and AVAILABLE, can be compared. Firstly, we might note that there is a correspondence between SEND TO REPAIR and REPAIR/MISSING. The correspondence is not exact but it appears that the former is included in the latter. This raises the issue of how books going missing are handled in the first diagram, and whether this needs to be represented. The actions BORROW and ISSUE appear to be identical, but note that the return action in one diagram is the inverse of issue, while in the other, it leads to a new state, ON RETURNED STACK. In this case we can assume that the state AVAILABLE in the second diagram is the composition of ON RETURNED STACK and ON SHELF. Other correspondences can similarly be found, and items may be involved in more than one correspondence.

The examples described demonstrate a number of different types of correspondence (Figure 7). Most obviously there is equivalence, as in the case of ISSUE and BORROW (Figure 7a). Often, different

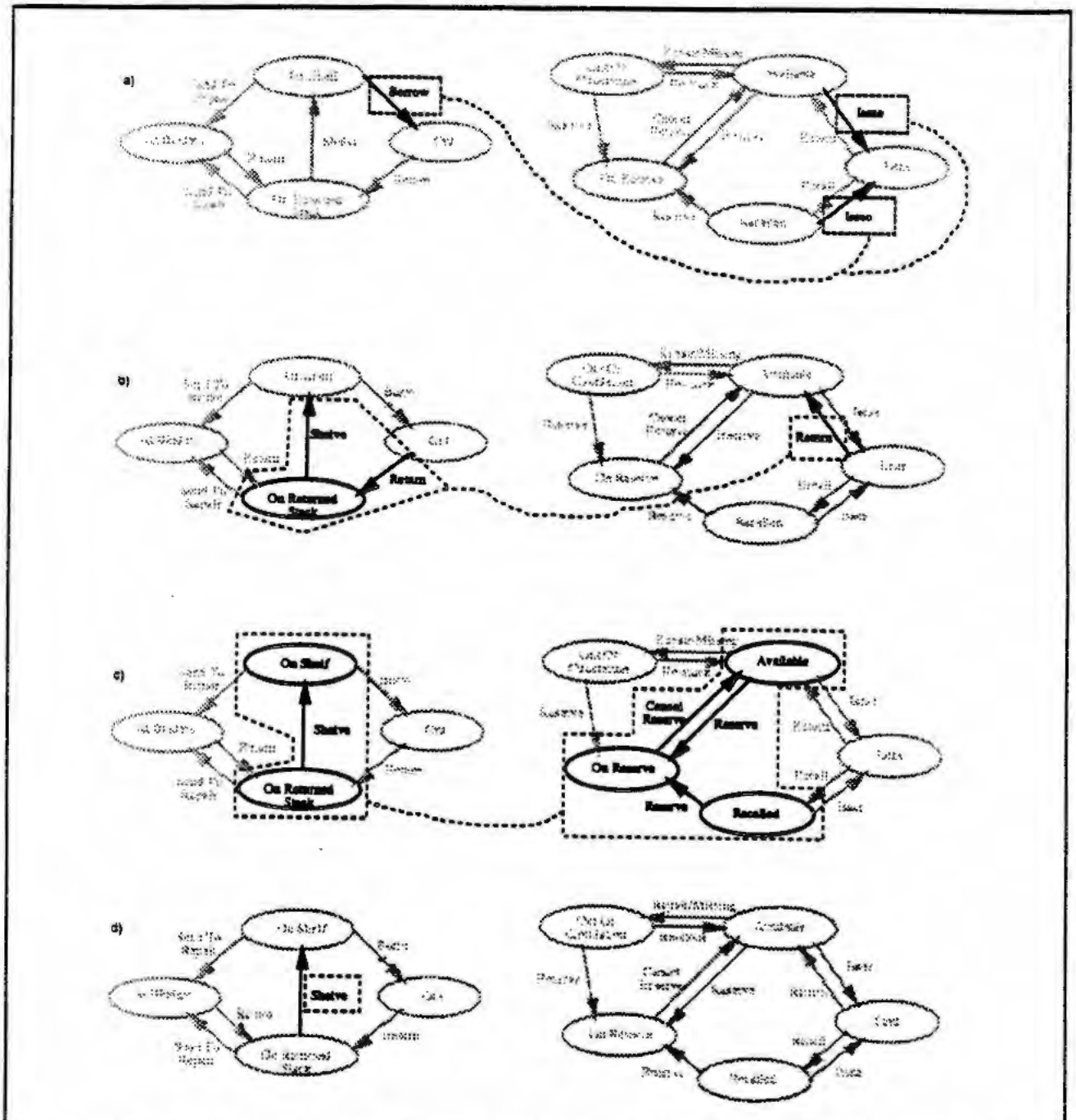


Figure 7: Correspondences between the descriptions of the library: (a) a correspondence between single items (although one of them recurs in the description); (b) a correspondence between a single item and a group; (c) a correspondence between two groups of items; and (d) an item for which there is no correspondence.

terms will be used to describe the same thing from different perspectives. In this case, one is the name of an action described by a librarian, and the other the same action described by a borrower. Both terms are useful, and could be recorded as synonyms. The comparison raises the issue of which should be used where.

As well as correspondences between single items, frequently groups of items will be linked. Where a single item in one description corresponds to a group in another, the representations are at different levels of decomposition (Figure 7b). This is a common problem in systems analysis, as there is no standard way of deciding whether different parts of a description are at the same level of

abstraction. The measure of level of abstraction is a subjective one, and different analysts will decompose the parts of a description in different orders. In many cases, such correspondences will not be exact, as decomposition usually reveals details about a description not considered at a coarser grain. Again, such comparisons yield issues that one description may not have addressed, which could be usefully discussed.

Correspondences between a group of items in one description and a different group of items in another description reveal where different types of decomposition have taken place. For example, the states ON SHELF and ON RETURNED STACK in the first description correspond to the group AVAILABLE, ON RESERVE and RECALLED in the second (Figure 7c). In this example, both groups are decompositions of "In the Library". The two groups will not necessarily match exactly. For example, the RECALLED state seems to include recalled books both before and after they are returned, and so is not totally captured within the group ON SHELF / ON RETURNED STACK. Different decompositions reveal different concerns within the system modelling process, in the same way as Feather's parallel elaborations [Feather 1989].

Finally there is the case where an item or group of items in one description has no correspondence in the other. This may be because it has been omitted, or because the role played by such an item has been filled in other ways. For example, the SHELVE transition in the second description has no correspondence in the first (Figure 7d).

The result of this stage is a list of correspondences between items in the viewpoints. These can be divided into those where the correspondence is exact, and those where it is only partial. The former indicate where there is agreement, and so restrict the area of conflict. However, where the correspondence is partial, there is still conflict to be resolved. In effect, the initial rough description of the original conflict has been replaced with a list of specific conflicts between items in the descriptions.

3.2.2. Identifying the Conflict Issues

For a conflict to be constructive to the design process, the reasons the parties are in conflict must be ascertained. These reasons may vary from lack of communication and poor understanding of other parties, to differences between priorities and areas of concern [Robbins 1989]. Often the actual conflict is unrepresentative of the real issues which lead to the conflict. Deutsch [1973] calls these *displaced conflicts*, and discusses the psychological reasons which cause people to express conflict in indirect ways. Although Deutsch's considerations are mainly to do with conflicts in personal relationships, there are other reasons why conflicts may appear displaced. The most likely reason is that the descriptions being compared are the result of long chains of development which are not necessarily based on the same initial assumptions and motivations.

Easterbrook [1989] notes that simply asking people to state any assumptions made by their descriptions is unlikely to be fruitful. There will be many assumptions, goals, and motivations involved in any description, some very trivial, and only a few will be relevant to the analyst. They are idiosyncratic in that what is obvious to one person may be a major decision to another [Kaplan 1989]. Discussion of assumptions must be prompted in some way, by asking "Do you assume X?" rather than "What do you assume?".

The systems gIBIS [Conklin 1989] and Argnoter [Stefik *et. al.* 1987] introduced the notion of *issues*, which are simply points that the design needs to address. They may take the form of suggested requirements (e.g. "The check-out process should ensure the borrower has not taken too many books"), or questions which need to be resolved (e.g. "How many books is too many?"). However, in these systems, issues are elicited unprompted: in gIBIS as a prelude to identifying positions and in Argnoter as supportive arguments for proposals. Our approach is to elicit issues only as a response to specific conflicts. Conflict provokes discussion of the issues, as participants raise issues they feel others have neglected. This prompting avoids having to ask participants simply to list any assumptions their description makes. It also avoids time wasted discussing

issues on which there is already agreement, or which are irrelevant to the current context. Issues can be attached using hypertext-like annotations, both to items in the descriptions, and to the correspondences that have been established between them.

To assist with the elicitation of issues, we allow four types of annotation to be attached to the items in the descriptions, and to the links between items:

Comments - these are general purpose annotations, which can be attached to any item or group of items in the conflict. A typical use would be to attach to a correspondence between items to suggest a reason for the difference or similarity of items. E.g. A comment might be attached to the state ON RETURNED STACK noting "a returned stack is not modelled in librarian B's description".

Assumptions - these are like comments but indicate where a description makes some unstated assumption. These often arise when two descriptions are compared, and the comparison reveals issues that have been neglected in either description. E.g. An assumption may be attached to the comment above, to note that "librarian B's model assumes that books waiting to be shelved can be located for loan".

Issues - these are points that need to be addressed. There are many circumstances under which issues arise, but often comments and assumptions will result in an issue. E.g. the assumption above might lead to the issue: "How can books that have been returned but not shelved be traced?".

Justifications - These are added to support a particular viewpoint or proposal. Often these will be added in response to assumptions and comments to provide a rationale for the original item. They will also be added in the next two phases of the process, to relate solution components to issues.

Several of the examples in the previous section showed how issues arise during the comparison of descriptions. Typically, issues are prompted by the creation of a correspondence, and the supporting tools prompt the user to note any assumptions or issues that arise when creating a correspondence. Assumptions have an issue attached automatically, questioning whether the assumption is reasonable, to ensure that the assumption is discussed when the issues are considered later in the process.

3.2.3. Agreeing criteria for evaluation of solutions

The final part of the exploration phase is the establishment of criteria by which to judge possible resolutions. Fisher & Ury [1981] suggest that objective criteria should be agreed before any resolutions are generated, to ensure that an agreement can be reached. This is to prevent participants moving the goal posts to get their personal preference accepted. However, it is often not clear before any solutions are proposed what the criteria should be. It is an open question as to how the prior agreement of criteria affects the generation of a resolution.

The establishment of criteria is clearly part of the exploratory phase, involving elicitation of further information from the participants. The criteria are measures by which potential resolutions can be judged and compared. They are the measures by which participants evaluate their satisfaction, and so represent their goals for the conflict resolution process. As such, these are the final part of the picture of the conflict built up by the exploration process. Issues represent the key points in the conflict, and so criteria show how the participants feel about these key points.

In our model, every issue has a criteria attached describing desirable outcomes for that issue. In the simplest case, the issue asks a question, and the criteria attached to it provides an answer (often approximately), by reference to the participants' concerns. Effectively we treat issues as points that the original viewpoints left unclear, and the criteria as the clarification of these points. In most cases, the participants will agree on the criteria for an issue with little difficulty, reserving disagreements for the order of priority among criteria. However, there are issues on which participants will define opposing criteria, and in this case, both are recorded, and the dispute added to the list of items in conflict.

Criteria can be used by participants to object to an issue. As issues are elicited in response to

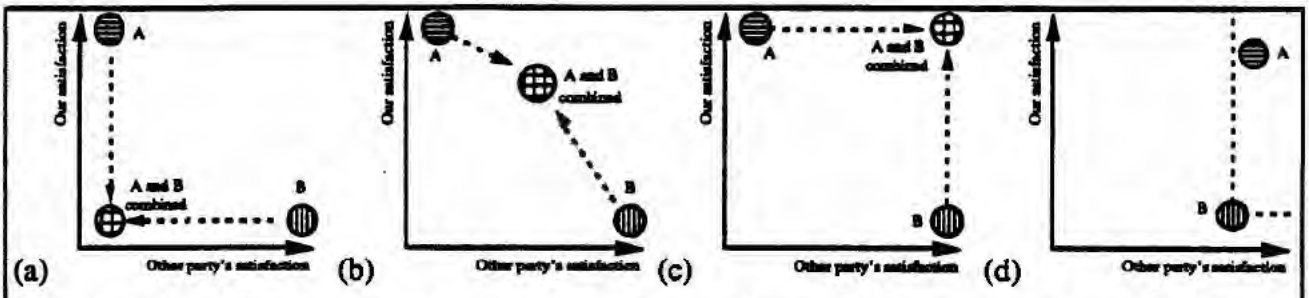


Figure 8: These diagrams show conflicts of different severity. In (a) the viewpoints are mutually exclusive, as their combination satisfies neither (the combination might not even be possible). In (b) the viewpoints can be combined, but with some loss of optimality for each party, and in (c) the viewpoints are non-interfering and can be directly combined. A variant of the non-interfering type is shown in (d), where one of the viewpoints already satisfies the other's concerns.

conflicts, they will usually be agreed as being valid, if only because they are important enough to disagree over. However, occasionally, participants will object to an issue on the grounds that it is not really an issue, or is irrelevant. In this case, they can attach a null criteria, effectively stating "this issue does not need to be satisfied (in my opinion)". Normally a null criteria will have either an assumption or a justification attached, explaining why. For example, a participant may object to the issue "There must be a way of locating unshelved books" because books are assumed to be unavailable until shelved.

3.3. The Generative Phase

The result of the exploration phase is a "map" of the conflict, which can be used to guide the search for possible resolutions. The second phase is concerned with generating these resolutions, and is essentially a design process. The aim is to propose a number of options which might overcome the limitations of the original viewpoints, and respond to the issues identified in the exploration phase. At this stage, the options are not evaluated, nor are they checked against the issues. This prevents the creative process being stifled by pragmatic considerations [Stefik *et. al.* 1987]. The options might be generated in a variety of ways, from directly combining elements of existing viewpoints to lateral thinking and brainstorming.

Before the generative process gets underway, it is useful to characterise the type of conflict as revealed by the exploration process. This will help to decide what form the generative phase will take, and what a possible resolution might consist of. We can identify three main types of conflict that might arise in systems analysis, as follows:

Conflicting interpretations - descriptions of the system as it is now, or its requirements, do not match. This corresponds to the category *Beliefs* (or "How things are"), as described by Deutsch [1973].

Conflicting designs - suggestions (or partial designs) for how the system should be do not match. This corresponds to the Deutsch's category *Values*, or "How things should be".

Conflicting terminologies - the terms in which things are described do not match. This covers the communication problems suggested by Robbins [1989] as being a major cause of conflict.

In addition to these three types of conflict we use a scale for the severity of the conflict. This ranges from *non-interference* at one end to *mutual exclusion* at the other. The former implies the items in conflict can be combined directly with no interference, whilst the latter indicates that each totally negates the other, and only one can be used (Figure 8).

Table 1 shows examples of each of the types of conflict, for different levels of severity. Using this schema, conflicts identified as non-interfering can be eliminated from further resolution work. A resolution readily apparent for these conflicts is the direct combination of the two viewpoints. Where the two viewpoints provide alternative views or alternative terms, the circumstances under which each should be used still needs to be examined. For the remaining conflict types, there is plenty of scope for the design of novel resolutions which circumvent the problems of the existing

viewpoints, by satisfying the issues underlying them in other ways.

3.3.1. Generating solution components

Our model does not prescribe a particular method of generating resolutions. As already noted, this is a design problem, and design methods are already well documented elsewhere [Finkelstein & Finkelstein 1983]. We do, however, provide one tool to assist in this process, which simply takes each issue in turn, and prompts for ways in which the issue can be satisfied. Each suggestion is then recorded as a potential resolution.

The division into conflicts between interpretations, designs and terminology helps to determine what form a resolution should take. For example, conflicts in terminology, once detected, can be resolved fairly simply. Participants are prompted for distinguishing terms when the same terms have been used for different concepts. Each such suggestion is a possible resolution, to be evaluated in the same way as other proposals. Where different terms have been used for the same concept, it is likely that both will be useful, and proposals will include circumstances under which a particular term might be favoured.

Conflicting interpretations are slightly harder to resolve. Sometimes these will be based on incorrect information, which can be investigated. More often, they will arise from alternative ways of looking at things. Again, both interpretations might be useful, and proposals can be made which attempt to combine them, or which suggest circumstances under which one or other might be used. Proposals might also recommend that one interpretation should be discarded, in which case the issues raised by the discarded description should be satisfied in other ways.

Conflicting designs involve a higher level of uncertainty. Often the conflict will be the result of conflicts not tackled at earlier stages, and the issues arising out of conflicting designs will indicate the concerns that lead to them. As the exploration stage has broken down the original conflict into a series of specific conflicts, the designs can be examined more closely. Possible resolutions include combining parts of the designs, adapting one design to incorporate issues raised by another, or making a totally new design which addresses the issues in new ways. Note the distinction between the descriptions involved in conflicting designs, and resolution options. While both are proposals of a sort, they are treated in different ways. Resolution options are effectively proposals for reconciling conflicting design proposals.

The result of this phase is a set of options for resolution. These will vary from the very specific (for example, a circumstance under which a particular term might be favoured), to entire viewpoints. Note that the original viewpoints could be considered as possible resolutions: one could be accepted unaltered in an either/or decision. In addition, some proposals will be candidates

	Conflicting Interpretations	**Conflicting Designs**	**Conflicting Terminology**
non-interfering	Either interpretation can be used without affecting the other (need to find out which to use when)	The design can be directly combined without compromising either concern.	different terms have been used for the same concept. Need to find out which to use when.
partially interferring	Interpretations are not wholly consistent, and if both are to be used, some resolution is required.	Designs can be combined but interfere, and the direct combination may not be the ideal resolution.	The same labels have been used for similar concepts. The differences need to be resolved
mutually exclusive	Interpretations totally contradict one another, and cannot be used in conjunction.	Designs are completely incompatible, or their direct combination produces a worse result than either.	The same labels have been used for different concepts, and some distinguishing terms are needed.

Table 1: Different types and severities of conflict.

for combination to produce a more complete resolution, while others will be combinations which might need to be dismantled, if only a part is needed. At this stage, the proposals have not been evaluated or compared in any way.

3.4. The Evaluation phase

The final phase, evaluation, consists of taking the options for resolution and relating them both to the map of the conflict generated in the exploration phase, and to each other. The aim is to find the option or combination of options that best resolves the issues involved in the conflict. The approach is similar to that of the exploratory phase, and consists of linking items together and eliciting extra information to supplement the links. Note that the generative process need not stop once the evaluation starts: the act of relating the options to the map may suggest new possibilities.

3.4.1. Relating the solution components to the issues in conflict

The first task is to relate the options for resolution to the issues underlying the conflict. There are two possible approaches to this task. Either we could take each option in turn, and decide which issues it satisfies, or we could take each issue in turn, and decide which options satisfy it. Both approaches have merit, in that either may reveal additional links missed in the other. Satisfaction of issues is measured using the criteria attached to them. The links between options and issues are elicited and recorded in a similar way to the correspondences in the first phase.

Links between options and issues can be positive or negative. The former indicates the option contributes to the satisfaction of the issue, while the latter indicates it frustrates the issue. In addition, such links can have a value attached to indicate the strength of the contribution. These can be used to compare options which contribute towards an issue. Options and links can be annotated with assumptions, justifications and notes.

3.4.2. Relating the solution components to one another

Once the options have been related to the issues, we need to find out how the options relate to one another. In some cases, options can be combined to enrich one another, while for others, combination will negate some of the benefits. The possible interactions are analogous to the conflict severities shown in figure 8. Indeed, if we reverse the scale of severity of conflict, we obtain a scale of compatibility.

There are two ways that interactions between options can be elicited. Firstly options can be directly compared and an annotated link created, recording their level of compatibility, and the circumstances (if any) under which they can be combined. Such links can be directly manipulated by the tools to show combinations as new options, and to re-compute the links with the issues. Secondly, the links between options and issues can be compared to discover the effect that interaction between options has on an issue. For example, two options which contribute towards an issue may combine to increase the level of satisfaction. Some such links may even assume that another option is also taken. Alternatively, two options may prevent one another from properly satisfying an issue.

3.4.3. Evaluation in terms of the agreed criteria.

Once the options have been linked to the issues and to each other, the only remaining problem is to select the best option or combination of options as a final resolution. For this evaluation process, the automated support presents the information acquired thus far as an intelligent knowledge base which can be interrogated in various ways. For example, the tool can list the options that are compatible with a particular option, and demonstrate how various combinations satisfy the issues. Given a priority ordering for the criteria, the tool can calculate the best combination of options. A number of different strategies for decision making can be used, some of which the tool knows about, while others can be programmed.

The chosen resolution is represented as a new viewpoint which can be used instead of the original

conflicting descriptions. The conflict map is recorded as a rationale for the resolution, to be re-examined if necessary.

4. SUMMARY

We have described a model of negotiation which can be used to support groups of people working to integrate their various perspectives. The model concentrates on the exploration of each other's perspective prior to entering a generative phase in which solutions to conflicts are sought. In this sense we combine the two most co-operative modes of conflict resolution: education and negotiation. Emphasis is placed on the exploratory phase in which participants learn about other viewpoints by comparing them to their own. During this phase, the initial conflict is broken down into a series of specific correspondences and conflicts between items in the viewpoint descriptions. These are annotated with comments describing any assumptions they make and issues they raise. The list of links and annotations builds up as a map of the conflict to guide the later stages.

Resolution takes the form of designing novel ways of satisfying the issues. The ideas generated are then compared with one another and measured against the issues to determine the level of satisfaction. Where ideas can be combined, new links are created to show the result of combination.

The entire process is highly interactive, and acts to structure the elicitation of additional information concerning the conflict. Rather than imposing a strict methodology on the participants, the tools provided can be freely interleaved, so as to support discussion and exploration. A number of conflict resolution strategies can be tested in this way, and bids can be represented by adding nodes and links to the hypertextual knowledge base. For example, a trading of concessions can be represented by linking the concessions together with a record of the trade. Challenges to issues in the various descriptions can be represented, and arguments given to support a position can be linked to that position. Where an agreement is not reached, the arguments built up aid judicial arbitration.

5. ACKNOWLEDGEMENTS

The author is indebted to Anthony Finkelstein, for patient and perspicacious supervision, and to Bill Robinson, for useful discussions on the ideas presented here. Much of the inspiration for this work came from a trip to the Knowledge Science Group at Calgary University, and special thanks are due to Brian Gaines for making this possible, and to all the other members of the group for providing food for thought. The work was funded by the SERC, studentship number 87311891.

6. REFERENCES

Anderson, J. S., and Fickas, S. (1989) *A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem*, Proceedings, Fifth IEEE International Workshop on Software Specification and Design, Pittsburg, Penn.

Axelrod, R. (1984) *The Evolution of Co-operation*, Basic Books Inc, NY.

Conklin, J. (1987) *A Survey of Hypertext*, Tech. Report No STP-356-86, Micro-electronics and Computer Technology Corporation (MCC), Austin, Texas. Reprinted in Greif, I., (ed) (1988) *Computer-Supported Co-operative Work: A Book of Readings*, Morgan Kaufmann, San Mateo, CA.

Conklin, J. (1989) *Design Rationale and Maintainability*, Proceedings of the Twenty-Second Annual IEEE International Conference on System Sciences, Vol 2, Hawaii.

Curtis, B., Krasner, H., and Iscoe, N. (1988) *A Field Study of the Software Design Process for Large Systems*, Communications of the ACM, 31 (11).

Davis, R. (1979) *Interactive Transfer of Expertise: Acquisition of New Inference Rules*, Artificial Intelligence 12, p121-157.

De Bono, E. (1985) *Conflicts: A Better Way to Resolve Them*, Penguin Books.

Deutsch, M. (1973) *The Resolution of Conflict*, Yale University Press, New Haven.

Easterbrook, S. M. (1989) *Distributed Knowledge Acquisition as a Model for Requirements Elicitation*, Proceedings, Third European Workshop on Knowledge Acquisition for Knowledge Based Systems (EKAW-89), Paris.

Feather, M. S. (1989) *Constructing Specifications by Combining Parallel Elaborations*, IEEE Transactions on Software Engineering, 15 (2), Feb 1989, p198-208.

Finkelstein, A. C. W., Goedicke, M., Kramer, J., and Niskier, C. (1989) *ViewPoint Oriented Software Development: Methods and Viewpoints in Requirements Engineering*, Proceedings, Second Meteor Workshop on Methods for Formal Specification, Springer-Verlag, LNCS..

Finkelstein, A. C. W., Maibaum, T. S. E., and Finkelstein, L. (1989) *Engineering-In-The-Large: Software Engineering and Instrumentation*, Draft Report, Imperial College London.

Finkelstein, A. C. W., and Fuks, H. (1989) *A Co-operative Framework for Software Engineering*, Proceedings. Twenty-Second Annual Hawaii International Conference on Systems Sciences, Vol 2, p189-199.

Finkelstein, L., and Finkelstein, A. C. W. (1983) *Review of Design Methodology*, IEE Proceedings, 130 (4), June 1983.

Fisher, R., and Ury, W. (1981) *Getting to Yes: Negotiating Agreement Without Giving in*, Hutchinson.

Greenberg, S. (1989) *A Survey of Computer-Supported Co-operative Work*, Draft Report, Alberta Research Council, Calgary, Canada.

Huhns, M. N. (ed) (1987) *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers Inc, Los Altos CA..

Kaplan, S. M. (1989) *COED: Conversation-Oriented Software Environments*, Draft Report, University of Illinois at Urbana-Champaign.

Keeney, R. L., and Raiffa, H. (1976) *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, J. Wiley & Sons, NY.

Lehman, M. M. (1990) *Uncertainty in Computer Application*, Technical Correspondence, Communications of the ACM, 33 (5), May 1990.

Lowe, D. G. (1986) *SYNVIEW: The design of a system for co-operative structuring of information*, Proceedings, Conference on Computer-Supported Co-operative Work, Austin, Texas, p376-385.

Luce, D. L., and Raiffa, H. (1957) *Games and Decisions: Introduction and Critical Survey*, J. Wiley & Sons, NY.

Nii, H. P. (1986) *Blackboard Systems (part 1): The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures*, The AI Magazine, Summer 1986.

Patchen, M. (1970) *Models of Co-operation and Conflict: A Critical Review*, Journal of Conflict Resolution, 14, (3), Sept 1970.

Rapoport, A., (ed) (1974) *Game Theory as a Theory of Conflict Resolution*, D. Reidel Publ. Co., Dordrecht, Holland.

Robbins, S. P. (1974) *Managing Organizational Conflict: A Non-traditional Approach*, Prentice Hall, NJ.

Robbins, S. P. (1989) *Organizational Behaviour: Concepts, Controversies, and Applications*, (fourth edition) Prentice Hall, NJ.

Robinson, W. N. (1990) *Negotiation Behaviour During Multiple Agent Specification: A Need for Automated Conflict Resolution*, To appear, ICSE-90.

Rosenschein, J. S. (1985) *Rational Interaction: Co-operation Among Intelligent Agents*, Ph.d. Thesis, Report No STAN-CS-85-1081, Dept of Computer Science, Stanford University, Stanford, CA..

Rosenschein, J. S., and Genesereth, M. R. (1985) *Deals Among Rational Agents*, Proceedings, Ninth International Joint Conference on Artificial Intelligence, p91-99.

Schuler, D. (1988) *AI and Hypertext in Support of Negotiation*, in Bernstein, M., (ed) (1988) Proceedings, AAAI-88 Workshop on AI and Hypertext: Issues and Directions .

Scott, B. (1988) *Negotiating: Constructive and Competitive Negotiations*, Paradigm Publishing, London.

Shaw, M. L. G., and Gaines, B. R. (1988) *A Methodology for Recognising Consensus, Correspondence, Conflict, and Contrast in a Knowledge Acquisition System*, Proceedings, Third Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, November 1988.

Shaw, M. L. G., and Woodward, J. B. (1989) *Mental Models in the Knowledge Acquisition Process*, Proceedings, Fourth Knowledge Acquisition For Knowledge-Based Systems Workshop, Banff, October 1989.

Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., and Suchman, L. (1987) *Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings*, Communications of the ACM 30 (1).

Strauss, A. (1978) *Negotiations: Varieties, Contexts, Processes and Social Order*, Jossey-Bass Publishers, San Francisco, CA.

Thomas, K. (1976) *Conflict and Conflict Management*, in Dunnette (ed), Handbook of Industrial and Organizational Psychology, Rand McNally College Publ. Co.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Meta-Tool Support for Customized Domain-Oriented Knowledge Acquisition

Henrik Eriksson

Department of Computer and Information Science
Linköping University, S-581 83 Linköping, Sweden
E-mail: her@ida.liu.se

Abstract

Meta-level environments is an approach to specialization of knowledge acquisition tools. DOTS is a meta-tool for the implementation of model-based knowledge acquisition tools. Once a conceptual model of the domain has been established, knowledge engineers can use DOTS to specify and generate a knowledge acquisition environment for developing expert systems directly by domain experts. DOTS assumes that the target tools are *model-based*, i.e. designed according to the expert's conceptual model of the domain.

A unique property of DOTS is that no particular domain or generic task is assumed for the target tools. This generalization is achieved by introducing an *abstract architecture*, i.e. an abstract model of the target tool, as the framework for specifying target knowledge acquisition tools. DOTS provides facilities for editing this abstract architecture and instantiating knowledge acquisition tools from it.

Manual implementation of model-based knowledge acquisition tools has previously been a focus of our work. DOTS is currently being evaluated by applying it to several domains in different generic tasks, including troubleshooting of laboratory equipment and experiment planning in biochemistry.

1 Introduction

It is possible to classify knowledge acquisition (KA) tools according to their scope with respect to the target domains and generic tasks they address[1]. The trend in the past decade has been towards more specialized KA tools. *Domain-oriented* KA tools represent the most restricted scope in this sense—one domain or even one application[2] (Eriksson, 1989; Gale, 1987; Musen et al., 1987). Although the generality of such tools is low, they can provide very efficient knowledge acquisition support. Naturally, the major impediment for the domain-oriented approach is domain and generic task dependency—much effort is required to implement tools for each domain.

[1]Throughout this paper, the term *domain* is used to denote an area of expertise (e.g. how to troubleshoot a specific device), whereas the term *generic task* (Chandrasekaran, 1986) denotes basic combinations of knowledge structures and inference strategies that are powerful for solving certain kinds of problems.

[2]The term *application* is used in this paper to denote a single expert system. One application does not necessarily cover an entire domain.

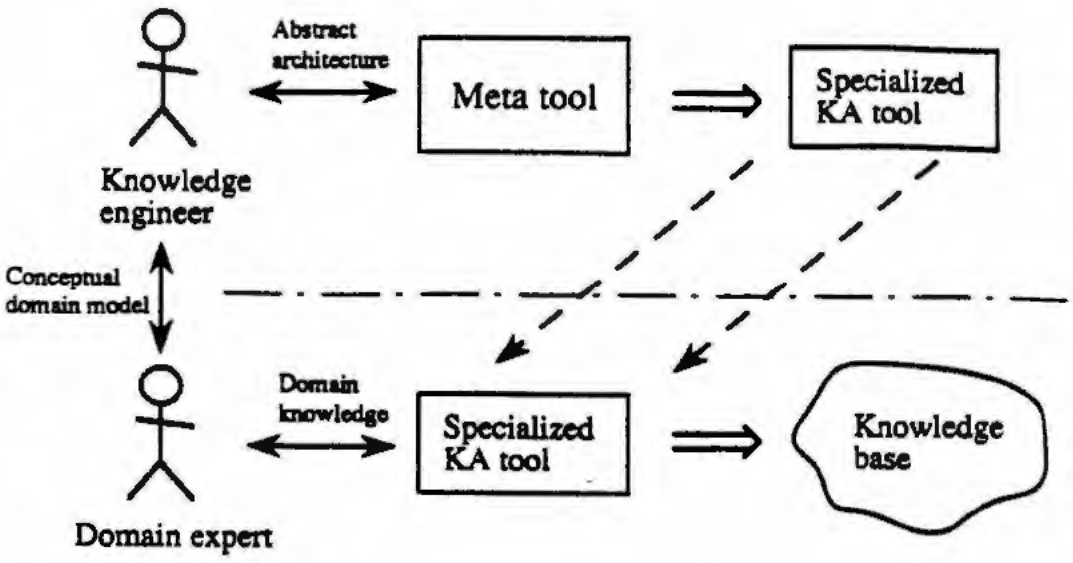


Figure 1: *Using a meta-tool to create specialized (domain-oriented) knowledge acquisition tools.*

One way of making the domain-oriented approach feasible is to utilize *meta-level environments* to support the realization of specialized and customized KA tools (see Figure 1). Thus, the meta-level provides the domain independence. The hypothesis is that utilization of a meta-tool to create specialized KA tools, which in turn are used to acquire knowledge for target knowledge bases, is less costly compared to applying a more general (e.g. generic task-specific) KA tool. This may or may not hold for a certain domain and size of the knowledge base. Moreover, meta-tools provide increased quality of acquired knowledge through personal KA environments for each expert.

This paper presents DOTS, a meta-level environment implemented by us that can be used by knowledge engineers to implement domain-oriented KA tools. A unique aspect of DOTS is its generality with respect to the target KA tools. Although the KA tools that DOTS generates are domain-, model-, and generic task-specific, DOTS itself does not rely on such assumptions. However, tools generated are restricted to *model-based* ones (Eriksson, 1989; Gale, 1987; Klinker et al., 1987; Musen et al., 1987; Musen, 1989). Model-based KA tools are based on the expert's conceptual model of the domain[3] or on a conceptual model shared by a group of experts. In model-based KA tools, experts enter their knowledge according to the model of the domain knowledge supported. Since model-based tools present models familiar to the experts, such tools usually rely on active entry of knowledge, for instance by form filling. Conceptual domain models are usually elicited before the KA tool is developed.

DOTS assumes that a conceptual model of the domain is established before the target KA tool is specified. This conceptual domain model can either be elicited by human interaction or through appropriate tools. Although DOTS does not directly support domain conceptualization, it can be used in this process by generating successive versions of the KA tool and thus aid in incremental development of the conceptual

[3]The term *conceptual domain model* (or conceptual model of a domain) will be used to denote a mental model of the domain including the nature and organization of knowledge in this domain (see Section 2.1). Note that different experts may have different conceptual models of the same domain.

domain model. Feedback from the domain expert can influence the next generation of the KA tool.

To produce a model-based KA tool with DOTS, knowledge engineers need to specify a model, or abstract architecture, of the target KA tool. In this process knowledge engineers are required to (a) tailor the user interface of the target tool, (b) declare the knowledge representation to be used in the KA tool, (c) specify the relationship between the user interface and the knowledge representation, and (d) formulate and enter transformation rules which map the knowledge representation into the final knowledge base structures. Through a set of menus and graphs knowledge engineers can enter and edit such descriptions according to a model of their desired target KA tool. From the abovementioned specifications, DOTS generates a KA tool that can be used directly by domain experts. Results from manual development of model-based KA tools provided the basis for the DOTS approach (Eriksson, 1989).

This paper is organized as follows: Section 2 presents an abstract architecture for generated KA tools. Section 3 describes the DOTS environment and how knowledge engineers interact with it. The KA tool generation process and the implementation of DOTS are presented in Section 4. Limitations of this approach and related work are discussed in Section 5 and 6 respectively. We conclude with a summary and discussion in Section 7.

2 A model of target KA tools

Before it is possible to discuss DOTS in detail, it is necessary to explain the underlying principle according to which knowledge engineers specify target KA tools in DOTS. This section presents a model, called *abstract architecture*, for the target KA tools. The conceptual model the user interface of DOTS presents to its users depicts this abstract architecture. The way users (i.e. knowledge engineers) interact with DOTS and edit the abstract architecture is further developed in Section 3.

Musen (1989) has taken another approach in the meta-tool PROTÉGÉ. PROTÉGÉ does not have any explicit model of the target tools it generates. Knowledge engineers specify properties of the domain with respect to the generic task supported (skeletal plan refinement) and PROTÉGÉ uses an *a priori* design, parameterized by the domain properties, for the target tools.

2.1 Conceptual domain models

A *conceptual domain model* is a mental model of the domain and in particular a *model of the knowledge* an expert uses for problem solving. A common domain model is needed for domain experts to communicate with each other, and some form of conceptual domain model is needed to communicate knowledge to a knowledge engineer and ultimately to a knowledge-based system (Regoczei and Plantinga, 1987). Consequently, knowledge acquisition involves elicitation of conceptual domain models. This elicitation can in principle be done in many ways, for instance through *knowledge level analysis* (Newell, 1982) or *ontological analysis* (Alexander et al., 1987).

A *model-based* KA tool exploits a conceptual domain model of a particular domain to acquire knowledge from experts (Eriksson, 1989; Gale, 1987; Klinker et al., 1987; Musen et al., 1987; Musen, 1989). The model a model-based KA tool presents to

its users (i.e. domain experts) should resemble their own conceptual domain model. However, the model supported by a model-based KA tool is *not* the same (and should not be the same) as the actual conceptual domain model. Most notably, the dialogue structure and discourse for the tool need to be considered in the tool model.

2.2 An abstract architecture approach

An important issue for meta-level environments is the question of specification style for the target KA tool. This is almost the same question as: What "meta-view" should be used? The approach taken in PROTÉGÉ (Musen, 1989) offers the user an abstract model of the generic task (skeletal plan refinement). Thus, the meta-view is a model of the domain at a higher, more abstract level, and a tool is generated from a description of this kind of the domain.

Our starting-point for target KA tool specification in DOTS is radically different. Since our aim is to achieve a general meta-tool which generates domain-specific KA tools, our approach is to move away from a generic task-oriented meta-view. Instead of trying to transform a description of the nature of the *domain* into a suitable tool, a meta-view focusing the target KA *tool* is utilized. PROTÉGÉ automates the transformation of the domain model into (an implicit model of) a KA tool. However, our principal view is that humans perform this transformation better and that specification according to a meta-view of target KA tools gives knowledge engineers much more freedom in designing the tools desired.

To specify a model-based KA tool we utilize a model of it. This model will be referred to as an *abstract architecture* of the target KA tool. The abstract architecture is *not* the same as a conceptual domain model, it should be regarded as a design of the target KA tool. An abstract architecture includes specifications of domain-related components as well as user interface, internal representations, logistics, and other similar aspects. Note that the abstract architecture should be regarded as a model or design for target KA tools; actually implementing them is the task of DOTS. Further, DOTS itself has no knowledge of what the target KA tool is eventually going to be used for, i.e. the knowledge that will be entered in the target KA tool. Because DOTS is a meta-tool, it has instead knowledge about the abstract architecture and the implementation of target KA tools.

The abstract architecture comprises *knowledge editors* and *knowledge modules*. The major components of the abstract architecture and relationship between them shown in Figure 2. Since the abstract architecture is intended for model-based tools, the knowledge editors are part of the user interface and handle knowledge editing with respect to the conceptual domain model. The knowledge modules are used for representing and storing knowledge entered through the knowledge editors. The domain knowledge is typically divided into knowledge "chunks," corresponding to the knowledge modules.

The motivation for this separation into knowledge editors and modules is (a) the desire to separate the surface interface from the deeper knowledge structures; (b) the fact that several editors can edit the same module, giving the user different "views" of the knowledge; and (c) the possibility to persistently store entered knowledge in a convenient way. The following sections describe in detail the knowledge editors and modules as well as update and transformation rules in. An alternative, vertical, view of the abstract architecture will be presented in Section 2.7.

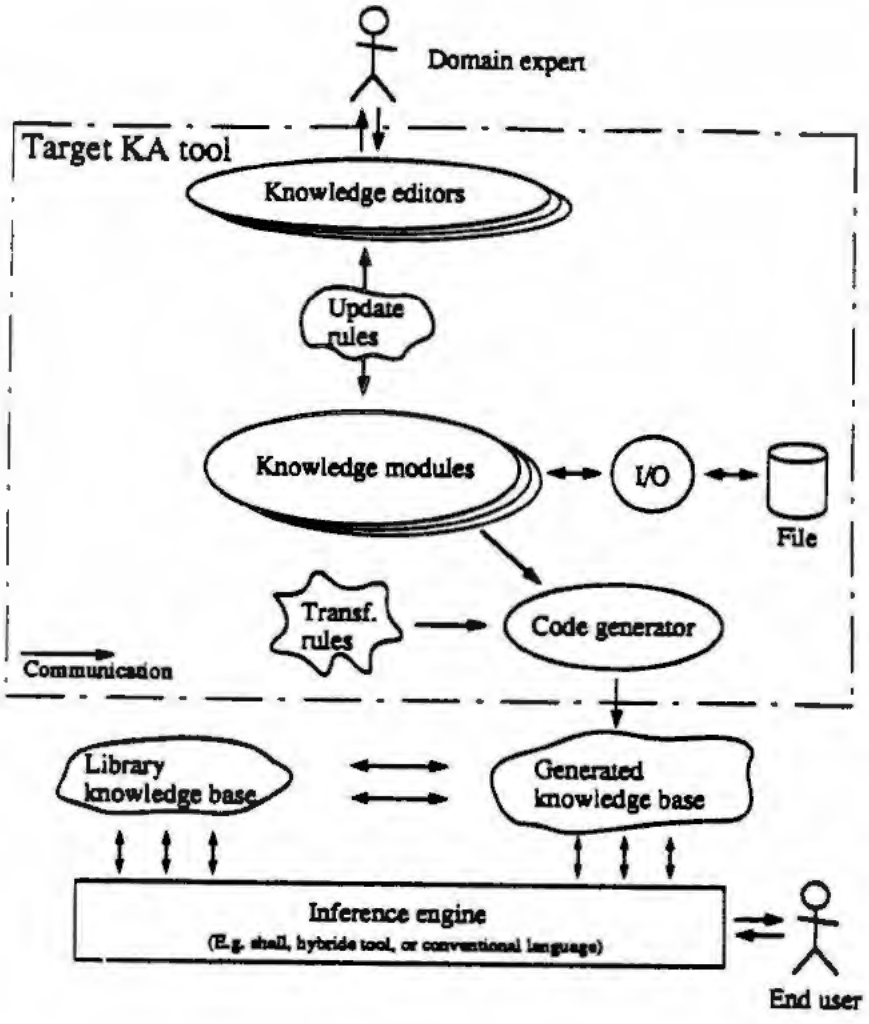


Figure 2: *Components of the abstract architecture of generated KA tools.*

2.3 Knowledge editors

The knowledge editors are a part of the user interface (Figure 2) in the target architecture. Their usage ranges from simple tasks such as handling menus and forms, to more complex tasks including graph editing and knowledge module browsing.

There are several different *types* of knowledge editors. These types are hierarchically organized with the type "knowledge editor" as the most general one. A number of built-in, or *basic*, knowledge editor types are also predefined in the model. These basic knowledge editor types serve as templates for the configuration of new types as well as a starting point for further specializations. New knowledge editor (sub) types can be introduced. Specific instances of knowledge editors can either be defined manually or created dynamically during runtime.

A set of operations (e.g. open, close, and edit) for each basic knowledge editor type is predefined. These operations provide a uniform interface to the window system and other machine-dependent functions which differ amongst the basic knowledge editor types. Usually, users of DOTS no not need to pay attention to such operations when defining new knowledge editor types since the appropriate operations are inherited (from

the basic knowledge editor type).

Whereas the model of the target tool's user interface resembles a hierarchy of editors, the actual implementation of the knowledge editors is based on object-oriented programming. Each class corresponds to a knowledge editor type and each instance corresponds to a knowledge editor instance.

An example of a knowledge editor is the *menu tool*, which is used to display menus to the user. DOTS has facilities for defining new specializations to the menu tool and functions for editing the menu layout (see Section 3.2 and Figure 7). Window positions, attached menus, icon layouts, etc., can also be specified.

2.4 Knowledge modules

The architecture utilizes *knowledge modules* for storage of "knowledge chunks," or knowledge-bearing units (see Figure 2). DOTS itself has no knowledge about the the meaning (with respect to the domain) of what is going to be stored in the knowledge modules of a target KA tool. Since DOTS operates on the meta-level, it has instead knowledge about the meaning of a *knowledge module* in a target KA tool architecture. Although not implemented in DOTS, meta-tools could be used to *define* semantics (with respect to the domain) for knowledge modules in target KA tools.

One of the motivations for the knowledge modules is to keep different kinds of knowledge (e.g. rule-, graph-, table-, and procedure-oriented) apart and to manipulate knowledge in a piecemeal fashion. When the expert has finished entering knowledge into a knowledge editor, the corresponding knowledge module is updated through an update mechanism guided by update rules (see Section 2.5). Note that the representation in the editor and the module may differ. Like the editors, the modules are organized in an inheritance hierarchy. At the top is the "knowledge module" type and the next level of detail consists of basic modules which represent predefined structures for browsers as well as special modules which handle system-related functions (e.g. save, load, and knowledge-base generation).

The knowledge modules will serve as a starting point for the transformation of acquired structures to the target knowledge base. An object-oriented implementation of the modules is highly appropriate for the transformation rules used to generate the target knowledge base (see Section 2.6).

2.5 Update rules

Consistency between the knowledge editors and the knowledge modules is maintained through a set of *update rules* (see Figure 2). These update rules map the contents (i.e. slot values) of the knowledge modules into knowledge editors and vice versa. In most cases this mapping is straightforward. However, in rare cases simple transformations might be necessary. The mapping declared through the update rules is inherited by specializations to the knowledge editors and modules. Further update rules might add more mappings between (slots of) the specializations. Update rules can be entered and edited in DOTS (see Section 3.3 and Figure 8).

2.6 Transformation rules

The target architecture relies on *transformation rules* (Figure 2) to transform the contents of the knowledge modules into target knowledge base structures. The purpose of the transformation rules in the target KA tool is to prepare acquired knowledge in the knowledge modules—which is in a format given by the model supported—for an inference engine, i.e. by converting it into a knowledge base at the symbol level. This may include syntactic as well as semantic transformations[4] of the knowledge in the knowledge modules into a suitable format for the target knowledge base. Again DOTS itself has no knowledge about the meaning of such transformations with respect to the domain. DOTS has knowledge about transformation rules in the target KA tool.

An inference mechanism for these transformation rules is also part of the target tool architecture. A special form of transformation rules is utilized for the task of generating knowledge bases from acquired structures. The scope of such rules refers to the knowledge module type hierarchy, i.e. each rule is applicable to a knowledge module type and its sub-types. The enabling conditions of these rules are adopted to the knowledge modules. The conclusions may either be attribute-value pairs or text templates. The purpose of the text templates is to provide means to describe resulting forms in the target knowledge-base format. The use of these templates is mainly motivated by the clarity of transformation rules. To achieve the desired output from the KA tool, DOTS provides facilities for writing and editing such transformation rules.

Another strategy for generating the target representation is exploited in PROTÉGÉ (Musen, 1989), where a relational database is used as an intermediate representation between the user interface and the target knowledge base.

2.7 A vertical view

Knowledge editors, knowledge modules, update rules, and transformation rules form "horizontal layers" of the abstract architecture (cf. Figure 2). The knowledge editors represent a surface layer (at the top) communicating with the expert, whereas the set of transformation rules is a deep layer (at the bottom) generating the knowledge base.

However, in many situations there is motivation for clustering architectural components according to a *vertical view* where corresponding knowledge editors, knowledge modules, update rules, and transformation rules are grouped together. For instance, if the domain expert points out a missing item in a knowledge editor the knowledge engineer would have to update a chain of components (starting with the knowledge editor and ending in the transformation rule set). Pairs of knowledge editors and knowledge modules are often used together and can be grouped together with the corresponding update and transformation rules.

The vertical view of the abstract architecture is expressed in the form of a *conceptual structure*. The conceptual structure is a global, high-level view of the target KA tool. A sample conceptual structure is shown in Figure 3. The conceptual structure comprises groups, or *units*, of abstract architecture components and the relationship between them. Units are abstract modules in the target KA tool model which may contain sets

[4]Syntactic transformation may, for instance, include transformation of a object into the appropriate knowledge base syntax. An example of semantic transformation is conversion from rules to decision tables and vice versa.

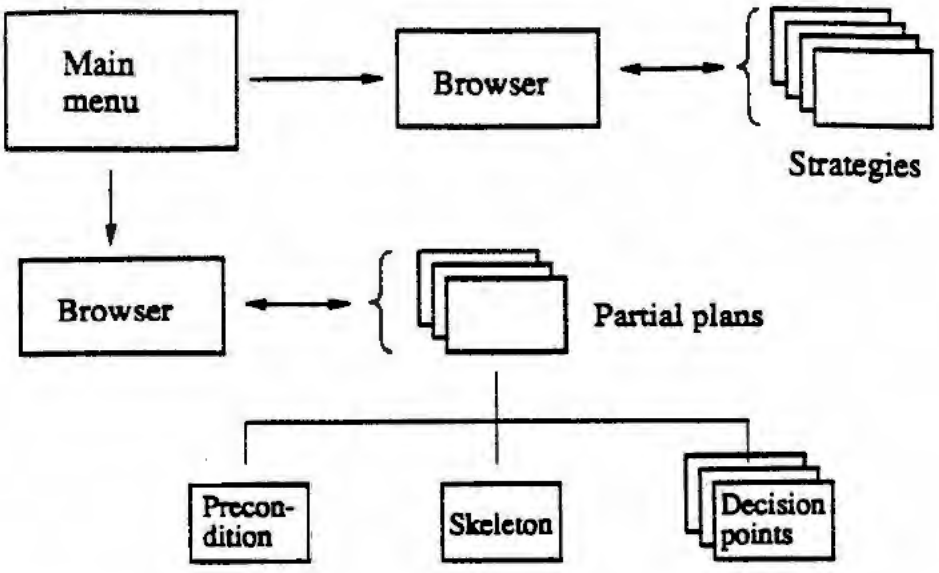


Figure 3: *A sample conceptual structure for a target KA tool. DOTS allows the knowledge engineer to edit a high-level conceptual structure reflecting the domain. The units in the conceptual structure (depicted as boxes) can be linked to other components of the abstract architecture (e.g. knowledge editors, knowledge modules, update rules, and transformation rules).*

of knowledge editors, knowledge modules, update rules, and transformation rules. In the approach taken the units may be linked to each other and to the abstract architecture objects they contain.

The conceptual structure is not mandatory for the specification of a target KA tool. Its primary purpose is to serve as a tool for the knowledge engineer to structure and organize a model-based KA tool on a high level.

2.8 Additional remarks

The model, or abstract architecture, of the target KA tool presented in Section 2 should *not* be seen as an actual architecture or implementation, but as a declarative *description* of the desired KA tool. DOTS supports this abstract architecture by offering this model (through its user interface) to the knowledge engineer. DOTS transforms the abstract architecture entered by the knowledge engineer into an efficient implementation of a KA tool for domain experts. Although the actual implementation of a KA tool is relatively close to the abstract architecture in DOTS, many alternative implementations are possible.

Naturally other models of KA tools are possible. In particular other views of target KA tools are possible, for instance, views focusing the "knowledge flow" (through the KA tool) from the domain expert to the knowledge base. The scope of DOTS, and thus this paper, is the abstract architecture presented.

There is, naturally, a limited family of KA tools which can be described in this architectural model. This kind of abstract architecture was primarily developed for model-based KA tools (Eriksson, 1989; Gale, 1987; Klinker et al., 1987; Musen et al., 1987; Musen, 1989) with a large proportion of active knowledge entry and editing by domain experts. Since DOTS is based on the abstract architecture presented, the class

of tools it can generate is restricted to those which can be described in these terms. This target architectural limitation is *not* the same as a restriction with respect to the generic task and the domain. However, model-based tools might be preferable for some types of applications. DOTS itself is an example of a tool implemented according to this abstract architecture.

3 The DOTS environment

DOTS is a meta-tool which can be used by knowledge engineers to develop and maintain model-based KA tools (see Figure 1). DOTS provides facilities for specifying KA tools according to the model (or abstract architecture) for target KA tools presented in Section 2. Knowledge engineers can interactively edit knowledge editors, knowledge modules, update rules, and transformation rules. The conceptual structure and the units can be edited in the same manner. Knowledge engineers can choose to specify the KA tool in any order. From the specifications entered, DOTS generates a KA tool that can be used directly by domain experts. The following sections deal with KA tool specification and editing in detail.

3.1 Defining the knowledge representation

Knowledge engineers can use DOTS to define the internal knowledge representation in the target KA tool. The basic building blocks for this representation are various types of knowledge modules (see Section 2.4). The knowledge module schema resembles a frame-based language with classes, slots, and instances.

The knowledge module classes (or types) and instances can be created and edited graphically. As shown in Figure 4, DOTS comprises two different browsers for knowledge modules: a linear one containing all user-entered knowledge modules and a graph browser showing the knowledge module hierarchy.

A number of knowledge module types are predefined. These basic knowledge module types are used for specifying the special functions in the target KA tool. These functions include browsing, knowledge base generation, loading from files, login functions, main menu functions, naming of new knowledge modules, pop up menu functions, and saving on files (cf. Figure 4). Depending on the nature of a knowledge module it is edited in different ways with specialized editors. The way a knowledge module is edited is inherited from a predefined parent. Knowledge modules handling special functions in the target tool require the specification and editing of properties that parameterize these functions. In addition to these specific properties of basic knowledge modules, the slots of knowledge modules types can be edited by a slot editor (see Figure 5). The knowledge engineer may indicate if the contents of each slot are to be saved between sessions, cached at start-up, and/or invalidated if the knowledge module is deleted.

Knowledge modules can be used to express decision tables, rules, etc., on the meta-level by letting various slots represent their contents.

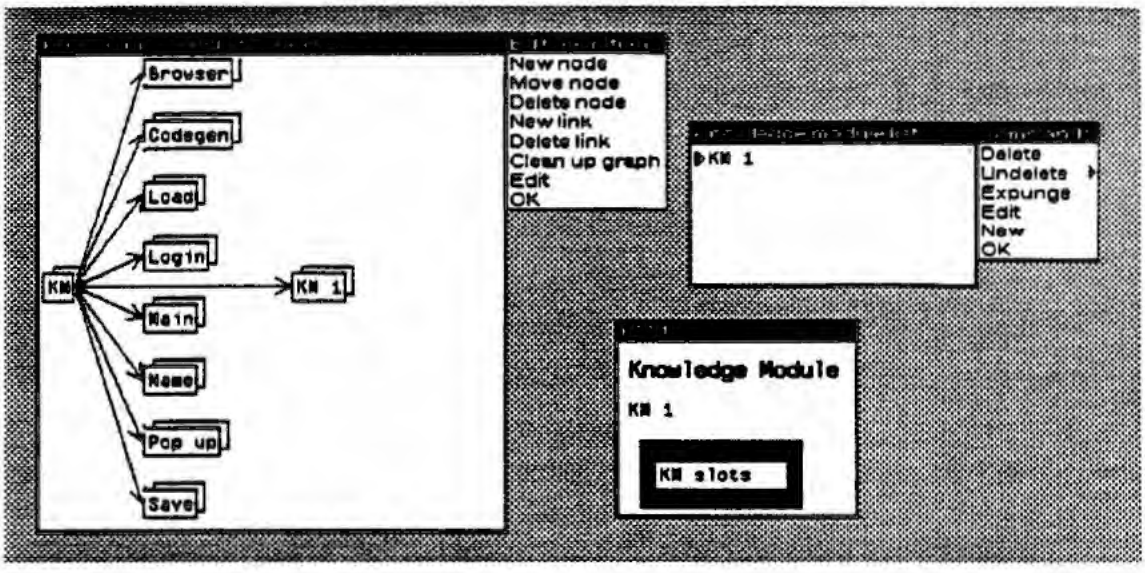


Figure 4: *Editing knowledge modules. The knowledge module taxonomy can be edited through a graph browser (left). The predefined knowledge module types are shown together with a user-defined knowledge module type (KM 1). There is also a linear browser showing user-defined knowledge module types (upper right). Since KM 1 is a plain knowledge module (KM), only the slots can be edited—through the "KM slots" button—in the editor for KM 1 (lower right).*

3.2 Tailoring the user interface

DOTS offers specialized UIMS[5] facilities for defining and editing the user interface of target KA tools. The knowledge engineer can tailor the user interface by defining such interface components as menu layouts, window properties, attached menus, etc.

The user interface of target KA tools consists of a combination of knowledge editors (see Section 2.3). The knowledge editor types and instances are organized in a hierarchy similar to the knowledge module taxonomy. User-defined slots are allowed in new knowledge editor types. As in the knowledge module case, there exists a set of basic knowledge editors. Many of them correspond to basic knowledge modules and are, thus, dedicated for editing them. DOTS comprises two browsers for the knowledge editors—a linear one and a graph browser for the hierarchy (see Figure 6).

A sample user-defined knowledge editor type (KE 1) is also shown in Figure 6. KE 1 inherits properties from the basic type "Freemenu," which is intended for defining freely formatted menus. The knowledge engineer may edit the menu layout as well as user-defined slots, window properties, and an optional attached menu. A screen picture from a session with the layout editor is shown in Figure 7. Such user interface components as push-buttons, radio-buttons, check boxes, text fields, etc., can be composed into a final menu by direct manipulation.

The knowledge engineer may link these user interface components to (user-defined) slots in the knowledge editor. Using these links, the status of components in the final menu (e.g. the contents of a text field) is automatically transferred back and forth

[5] User Interface Management System.

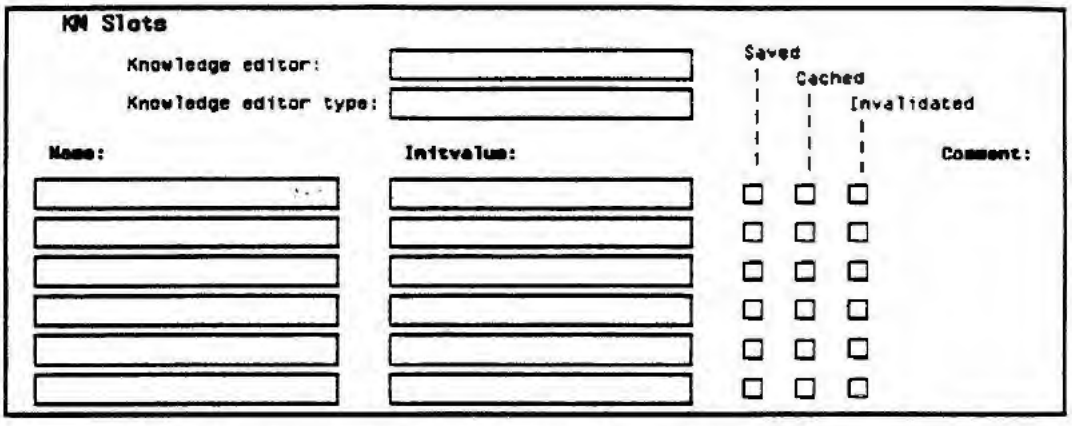


Figure 5: *Editing user-defined slots associated with a knowledge module type. Name and initial value as well as saved, cached, and invalidated facets of each slot may be specified. A specific knowledge editor or knowledge editor type can be specified for each knowledge module type.*

between the component and the corresponding slot of the knowledge editor. This value may in turn be transferred from/to the edited knowledge module by automatic update mechanisms in the target KA tool. These update mechanisms are driven by update rules (see Section 2.5 and 3.3).

FormIKA (Bennett, 1990) is a specialized textual language for the definition of similar menus. FormIKA is intended as an intermediate language for knowledge editors generated by PROTÉGÉ.

3.3 Defining internal relations

The relationships between the components of the target tool can be declared through update rules (see Section 2.5). The primary usage of the update rules is to define the relation between the knowledge modules (the deep structures) and the user interface in form of knowledge editors (the surface structures). The update rules declare the transfer of slot values from a knowledge editor into the corresponding knowledge module slots and vice versa. The mapping direction for each slot may also be specified. The update rules are inherited with respect to the knowledge editor taxonomy and the knowledge module taxonomy.

The knowledge engineer can create and browse through update rules using an update rule browser. The individual update rules can be edited by a specialized editor[6] (see Figure 8).

3.4 Specifying the knowledge base generator

The *knowledge base generator* in the target KA tool is the mechanism which transforms the contents of the knowledge modules into a knowledge base. This process is driven by a set of transformation rules (see Section 2.6). In other words the knowledge base generator is parameterized by the transformation rules. Thus, by specifying transformation

[6]Internally a condition (i.e. a premise) for each update rule can be specified. However, this facility was rarely used and was consequently removed from the update rule editor.

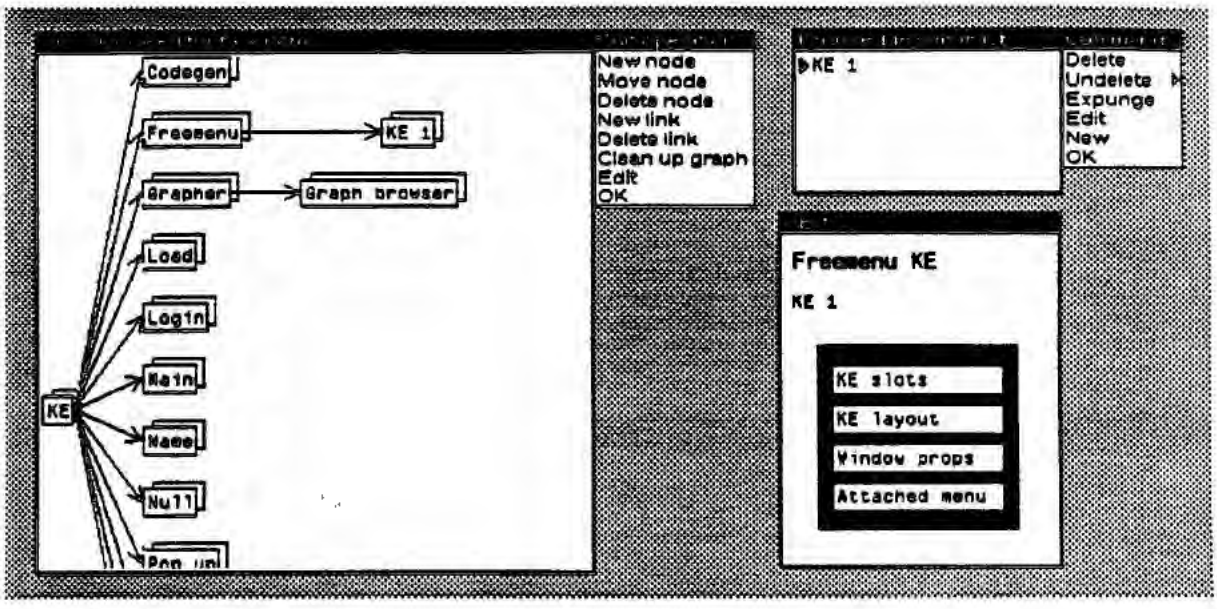


Figure 6: *Editing knowledge editors. A graph browser is used to edit the knowledge editor taxonomy (left). A sample user-defined knowledge editor type (KE 1) is shown in the knowledge editor taxonomy (together with basic ones) and in the linear browser (upper right). An editor comprising sub-editors for slots, menu layout, window properties, and attached menu is used to specify the knowledge editor type (lower right).*

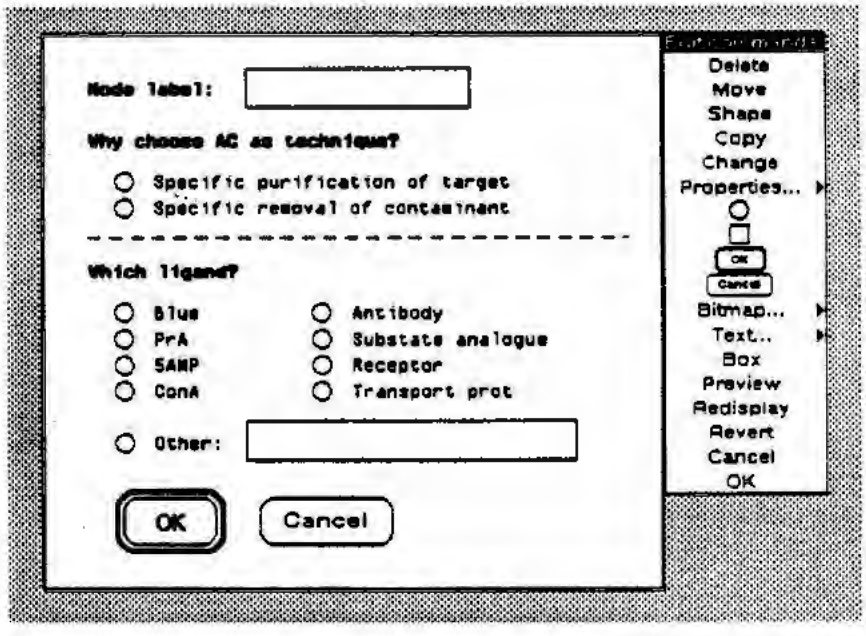


Figure 7: *Editing the layout for a menu knowledge editor. This example is taken from a biochemical domain.*

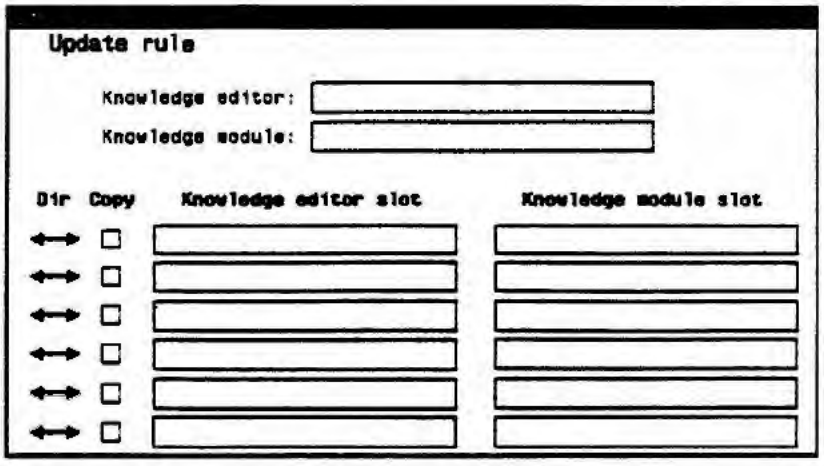


Figure 8: *Editing update rules. The knowledge editor and knowledge module types this rule applies to are entered at the top. Slots to be updated between knowledge editors and knowledge modules are subsequently declared. The direction of the update may be specified for each slot (bidirectional, knowledge editor to knowledge module, or vice versa).*

```
if x isa KM_1 and x.reason = :purification_of_target then

create rule

    "if current_node = {x.node_label} then
     conclude ligand = {x.ligand}"
```

Figure 9: *A sample transformation rule. (Transformation rules are used to transform the internal structures, i.e. knowledge modules, of a model-based KA tool into the final knowledge base.) This simplified version of a transformation rule transforms the knowledge in KM 1 into a rule in the target knowledge base.*

rules, the knowledge engineer can obtain the desired mapping from elicited structures into a suitable knowledge base format. Using this approach, knowledge base generators can be specialized for different target inference engines and different target languages by modifying the set of transformation rules.

A thorough treatment of knowledge transformation techniques for knowledge base generation is beyond the scope of this paper. However, we will provide an illustrative example of how knowledge modules can be transformed into production rules using this approach. Figure 9 shows a sample transformation rule suitable for mapping part of the knowledge contents of a knowledge module (of type `KM_1`) into a production rule. The premise of this rule is related to a knowledge module instance, whereas the conclusion is a production rule template. Runtime variable substitution instantiates a production rule (for the target knowledge base) from the template.

The rule in Figure 9 essentially states that: if x is of type `KM_1` and the "reason" slot of x is equal to `:purification_of_target` then the rule template

```
if current_node = {x.node_label} then
conclude ligand = {x.ligand}
```

will be instantiated. Assuming the slot "node_label" of x has the value N1 and the slot "ligand" has the value ConA the following production rule will be added to the target knowledge base:

```
if current_node = N1 then
conclude ligand = ConA
```

Note that this example is consistent with the layout of the menu knowledge editor KE 1, which is shown in Figure 7.

The set of transformation rules in the target KA tool can be edited in DOTS. Currently the transformation rules are expressed in a Lisp-like syntax in DOTS.

4 Generating KA tools

The knowledge engineer interacts with DOTS through a set of structure editors consisting of browsers, graphs, and menus. Using these editors, the target KA tool can be specified by the knowledge engineer according to an abstract architecture (see Section 2). The contents of these surface editors can be transformed into an internal representation in DOTS and back again. The internal contents, which define a target KA tool, can be saved on file for later modifications of the KA tool.

From the input to the KA tool generator in DOTS the abovementioned internal structures (originating from the user's specifications), the KA tool generator transforms these structures into efficient code for the target KA tool. The KA tool generation process in DOTS is based on a set of transformation rules. Generated code is linked together with a runtime library to form an executable KA tool. This library consists of core functions for window management, internal bookkeeping, file I/O, and knowledge base generation. In fact, many library routines are used by DOTS itself (e.g. window management function and file I/O).

Currently the primary bottleneck in gaining an executable KA tool is the time it takes to load and link generated code. Consequently incremental generation of KA tools was implemented in DOTS in order to facilitate an iterative development style. Non-complex changes to the abstract architecture can be propagated to the target KA tool with minimal delay.

The principles underlying the architecture of DOTS are quite similar to the target KA tool architecture schema (see Figure 2). DOTS itself can be described according to the abstract architecture presented in Section 2. Actually the DOTS implementation is metacircular, i.e. it is implemented in itself and is its own meta-tool. This implementation strategy facilitates maintenance and extension of DOTS. DOTS has been implemented in Common Lisp (Medley[7]) using CLOS[8] (Steele Jr., 1990).

PROTÉGÉ (Musen, 1989) uses a relational database for intermediate storage of information about the domain and specifications of graphical forms. An executable target KA tool is generated through a series database queries.

[7] Medley is a trademark of Venue.
[8] Common Lisp Object System

5 Limitations

Although DOTS is not limited to any particular application, domain, or generic task, it is limited to the type of KA tools it generates. This means that the major limitations for meta-tools supporting an abstract target tool model—such as DOTS—can be found at the meta-level, rather than at the level of final expert systems. In particular the limitations imposed by the nature of the abstract architecture restrict the target KA tool space. This restriction is to some degree flexible due to the metacircular implementation of DOTS since the target tool model supported by DOTS can be modified and extended by DOTS itself. A programmer's interface allows extensions to library functions in DOTS and target KA tools. For instance, new and fundamentally different types of knowledge editors can be implemented and introduced to DOTS by knowledge engineers.

Since the target KA tool space is restricted to model-based ones with a common basic architecture there are naturally limitations in DOTS inherited from this architecture (i.e. what the set of possible KA tools actually can do), for instance, limitations inherited from domain-oriented, model-based tools. DOTS can not, for example, generate KA tools which are based on expert interrogation in natural language, nor can it generate tools that are based on learning techniques.

6 Related work

PROTÉGÉ (Musen, 1989) is the principal work against which DOTS should be compared. Two other systems, KNACK (Klinker et al., 1987) and ASKE (Patel, 1989), will be compared to DOTS to illustrate KA tools with meta-level components and distinguish them from pure meta-tools such as DOTS and PROTÉGÉ. Before these systems are presented we will provide a classification schema for meta-level tools.

6.1 A classification of meta-level tools

At this point in the time only a few meta-tools—or tools exploiting meta-structures—have been reported. Nevertheless, it is possible to make a rough classification of existing ones and point out areas for potential new systems. We can distinguish between five important criteria:

First, a distinction can be made between approaches which separate tool support in two or more levels (e.g. meta- and KA tool levels) and systems that maintain both levels in the same tool. The DOTS approach is an example of an environment separated into meta- and target KA tool-levels.

Second, tools may or may not allow dynamic changes/refinements to the meta-level during KA sessions (interaction) with the domain expert. Such changes may deliberately be suppressed or not implemented as a consequence of design considerations (e.g. performance).

Third, the model of the meta-level may differ amongst the approaches. DOTS offers an abstract architecture as a model for target KA tools. Other approaches include *generic task-oriented* meta-tools, e.g. tools offering an abstract model of the reasoning strategy as the meta-level model, and potentially *domain-family-oriented* meta-tools—exploiting other similarities among domains—might be introduced. PROTÉGÉ (Musen, 1989) is an example of such a generic task-oriented meta-tool offering a conceptual model

Aspect	DOTS	PROTÉGÉ	KNACK	ASKE
Separate meta- and KA tools	Yes	Yes	No	No
Dynamically refined conceptual domain model	No	No	Yes	Yes
Meta-level conceptual model	Abstract tool architecture	Skeletal plan refinement	—	—
Assumed generic task	None	Skeletal plan refinement	Reporting tasks	Classification
Target KA tool family	Model-based	Model-based	Interview-oriented	Interview-oriented
Target inference engine	No assumptions	ONCOCIN	OPS5	Production rules

Figure 10: *A comparison between four tools with meta-components. (Note: DOTS and PROTÉGÉ are pure meta-tools, whereas KNACK and ASKE are KA tools which utilize a meta-level representation of the domain to guide the interrogation process.)*

of skeletal plan refinement to its users. Meta-tools may also assume a target generic task for KA tools delivered, e.g. classification, planning, etc.

Fourth, target KA tools (or in the case of an integrated meta-level—the tool itself) might adopt different knowledge acquisition strategies, for example model-based (as in DOTS and PROTÉGÉ) and interview-oriented[9] approaches.

Last, the target inference engine or target knowledge base format (language) might vary. Meta-tools may or may not assume a particular environment and/or format for the resulting knowledge base. The following section describes three systems related to DOTS.

6.2 Related systems

The related systems PROTÉGÉ, KNACK, and ASKE are briefly reviewed and compared to DOTS in this section. A summarized comparison according to the classification criterion in Section 6.1 can be found in Figure 10.

6.2.1 PROTÉGÉ

The closest related tool to DOTS is the PROTÉGÉ system, developed at Stanford University (Musen, 1989). PROTÉGÉ was developed as a meta-tool for OPAL, a KA tool used to acquire knowledge for the ONCOCIN system (cancer chemotherapy planning) (Musen et al., 1987). PROTÉGÉ generates model-based KA tools in the generic task of *skeletal plan refinement* (Tu et al., 1989). Knowledge engineers specify target tools

[9] I.e. oriented towards interrogation of domain experts in (restricted) natural language.

according to an abstract model of the generic task consisting of planning entities, actions, input data, etc. PROTÉGÉ instantiates model-based KA tools according to such specifications. Tools generated by PROTÉGÉ are intended to be used together with the ONCOCIN inference engine which interprets skeletal plans. Thus, PROTÉGÉ is a generic task-oriented meta-tool, limited to skeletal plan refinement and tailored to one target environment (ONCOCIN).

DOTS extends the meta-level support found in PROTÉGÉ by *not* assuming any particular generic task for the generated KA tools. DOTS-generated tools can be tailored for different knowledge base formats and inference engines by changing the set of transformation rules used for knowledge base generation. The gain of generic task-independence in DOTS is balanced by the requirement to specify *how* the target knowledge base should be generated (i.e. to write the transformation rules). This requires some work, but is not an unreasonable effort. Thus, the knowledge base generator of PROTÉGÉ's target KA tools is dedicated to one knowledge base format, whereas DOTS has a generalized knowledge base generator for the target KA tools. A different approach to generic task independence could be to generate task-based meta-tools from a task description. However, it is not clear how generic tasks and combinations of them should be specified.

In summary, the major advantage of PROTÉGÉ—compared to DOTS—is simple specification of the target tools. PROTÉGÉ generates target KA tools from a description of the domain with respect to its generic task. Although this simplifies specification, it is also a major drawback of PROTÉGÉ. Only KA tools for supported generic tasks, in the PROTÉGÉ case skeletal plan refinement, can be specified and generated. Different generic tasks might also require different knowledge acquisition strategies (Bylander and Chandrasekaran, 1987). Further, such aspects of the KA tool as menu layouts, user interface style, internal knowledge representation, and generated target knowledge base structures, is predetermined in PROTÉGÉ and can not be controlled and customized by knowledge engineers. The PROTÉGÉ approach is probably more suited for novice knowledge engineers than the DOTS approach, since the model of skeletal plan refinement in PROTÉGÉ gives the user some guidance in the specification task.

6.2.2 KNACK

KNACK (Klinker et al., 1987) is a KA tool which acquires knowledge for expert systems that assist with different reporting tasks. Through an initial interrogation session, which is driven by general knowledge about the reporting task, KNACK elicits a preliminary domain model. Domain models in KNACK describe concepts and vocabulary used by the domain expert. A sample report is also required as initial input. Guided by the domain model, KNACK interacts with the expert to match concepts in the domain model with fragments of the sample report. As a result of this process a generalized report is constructed. In the next stage, KNACK acquires the actual knowledge about how to customize the generalized report for a particular application. The preliminary domain model is also refined.

6.2.3 ASKE

ASKE (Patel, 1989) is an interrogation-base KA tool which elicits an meta-representation of domain concepts for guiding the interview. ASKE tries to combine the benefits

of general KA tools in the initial stage of knowledge acquisition with the advantage specialized tool support at later stages. ASKE is oriented towards classification tasks.

As in KNACK, the general idea in ASKE is to derive an initial task model through question-asking strategies and use this model to acquire the actual knowledge from the domain expert. During the initial question-asking session ASKE fills in a *general template* (containing domain classification and background knowledge for the project) and an *acquisition template* (containing general domain knowledge, including main concept types and their interrelationships). After the general domain knowledge (i.e. the conceptual domain model) has been elicited ASKE extracts the specific domain knowledge. This process is guided by the acquisition templates.

KNACK and ASKE are not directly comparable with pure meta-tools such as DOTS and PROTÉGÉ. KNACK and ASKE have some meta-level components integrated in the KA tool, whereas the DOTS and PROTÉGÉ approaches have separated the meta- and KA-levels in different tools intended for different users (i.e. knowledge engineers and domain experts). A summarized comparison is shown in Figure 10.

7 Summary and discussion

Separation of KA tool support into two levels of tools, meta- and target KA tools, can provide specialized KA tool support in a domain-independent manner. DOTS is a meta-level environment which allows knowledge engineers to create model-based knowledge acquisition tools. These KA tools may be tailored to a particular domain and even to a single expert. DOTS differs from other approaches to meta-level support by not presupposing any particular target domain or generic task for the KA tools it generates.

Another salient aspect is that the conceptual model of DOTS itself resembles an abstract architecture of a target KA tool. The knowledge engineer can use DOTS to (a) define the knowledge representation in the target KA tool, (b) tailor the user interface of the target KA tool by editing user interface components, (c) declare the relationship between the components in the target tool, and (d) specify the knowledge base generator of target KA tools by entering transformation rules that transform the internal (target KA tool) knowledge representation into a final knowledge base.

7.1 Discussion

So far the DOTS project has shown promising results in the use of an abstract model of the target KA tool as input to a meta-tool. Transformations from this abstract architecture into an efficient realization of a model-based KA tool can be implemented. Provided that a conceptual model of the domain can be established and an abstract architecture reflecting this model can be formulated, KA tools for a wide variety of domains and generic tasks can be generated by DOTS.

Development of a conceptual model of a domain is generally considered a key factor for knowledge acquisition. Gale (1987) suggests development of an expert system in the domain as a way of developing the conceptual domain model. Another approach is to build a model-based knowledge acquisition tool for the domain. We suggest the use of meta-environments to facilitate the elicitation of a domain model by incrementally developing a model-based KA tool. The expert should be able to run and comment upon several prototype versions of the KA tool, i.e. the conceptual domain model and the KA

tool should be developed *incrementally* and in *parallel*. Since the initial model is likely to require extensions and modifications during the process, meta-level environments should support modification of the conceptual domain model throughout the acquisition phase, and hence, modifications of the KA tool.

Since the DOTS implementation is metacircular it can be used to maintain and extend itself. An interesting prospect is that DOTS can potentially be used as a meta-meta-tool to implement other meta-tools, for instance generic task-oriented meta-tools (cf. PROTÉGÉ (Musen, 1989)). This would mean that meta-tools specialized to a class of domains (i.e. a class of target KA tools) or tailored to an organization's needs could be implemented with meta-meta-tool support.

7.2 Preliminary results

Getting a picture of the practical usefulness of the DOTS approach and verifying its generality would naturally needs extensive evaluation involving several applications in different domains and generic tasks. Currently, DOTS is being used to develop a model-based KA tool that acquires knowledge for troubleshooting laboratory equipment. Another test case involves planning of biochemical experiments. The aforementioned strategy for domain model elicitation is being applied in these projects with promising results.

We have previously studied manual implementation of model-based KA tools according to an earlier version of the abstract architecture and we expect results from this work to be applicable to KA tools generated by DOTS (Eriksson, 1989). Compared to manual implementation, considerable effort can be saved by applying DOTS. The performance of tools generated is comparable to hand-crafted ones.

At present the first version of DOTS has is implemented. We are currently evaluating this version by applying it to several domains in different generic tasks. In our continued effort we will use these results as input for further refinement of the DOTS environment.

Acknowledgments

Thanks are due to Peter Eklund, Anders Haraldsson, Sture Hägglund, Ivan Rankin, Kristian Sandahl, and Tomas Sokolnicki for valuable discussions and comments on drafted versions of this paper.

References

Alexander, J. H., Freiling, M. J., Shulman, S. J., Rehfuss, S., and Messick, S. L. (1987). Ontological analysis: An ongoing experiment. *International Journal of Man-Machine Studies*, 26(4):473–485.

Bennett, A. (1990). FormIKA: A form-based user interface management system for knowledge acquisition. Knowledge Systems Laboratory Report KSL-90-43, Stanford University.

Bylander, T. and Chandrasekaran, B. (1987). Generic tasks for knowledge-based reasoning: The "right" level of abstraction for knowledge acquisition. *International Journal of Man-Machine Studies*, 26(2):231–243.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30.

Eriksson, H. (1989). *A Study in Domain-Oriented Tool Support for Knowledge Acquisition.* Linköping Studies in Science and Technology, Licentiate thesis 181, LiU-Tek-Lic 1989:21, ISBN 91-7870-510-X, Linköping University.

Gale, W. A. (1987). Knowledge-based knowledge acquisition for a statistical consulting system. *International Journal of Man-Machine Studies*, 26(1):55–64.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M., and McDermott, J. (1987). KNACK—report-driven knowledge acquisition. *International Journal of Man-Machine Studies*, 26(1):65–79.

Musen, M. A. (1989). Conceptual models of interactive knowledge acquisition tools. *Knowledge Acquisition*, 1(1):73–88.

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(1):105–121.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

Patel, J. (1989). On the road to automatic knowledge engineering. In *Proc. of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI'89*, pages 628–632.

Regoczei, S. and Plantinga, E. P. O. (1987). Creating the domain of discourse: Ontology and inventory. *International Journal of Man-Machine Studies*, 27(3):235–250.

Steele Jr., G. L. (1990). *Common LISP: The Language.* Digital Press, second edition.

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(12):1439–1455.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

ICONKAT: An Integrated Constructivist Knowledge Acquisition Tool

Kenneth Ford[1], Howard Stahl[2], Jack Adams-Webber[3], Joseph Novak[2], Alberto Cañas[1], Jeremy Jones[1]

ABSTRACT

We report on a continuing research effort aimed at the development of an integrated knowledge acquisition system, ICONKAT. We describe the components of the tool and discuss how they may be used to facilitate the design, construction, testing and maintenance of knowledge bases. ICONKAT's knowledge elicitation subsystem, based on both Personal Construct Theory and Assimilation Theory, assists in the elicitation of a domain expert's object space as well as the rules that he/she employs to reason over that structured space. Thus, ICONKAT directly supports knowledge acquisition for the recent hybrid AI performance environments (e.g., NEXPERT™) that employ both representation and reasoning dimensions. This tool is currently employed in the design and construction of an expert system for the diagnosis of first pass functional cardiac images.

1. INTRODUCTION

The first, and most fundamental, step in the knowledge acquisition process is the elicitation of human expertise and its subsequent constructive modeling in a computer system. The mining analogy notwithstanding, expertise is not like a natural resource that may be harvested, transferred, or captured, but rather it is constructed by the expert and *reconstructed* by the knowledge engineer. A pivotal aspect of the knowledge acquisition process is the knowledge engineer's act of construction in which the utterances/tool-interactions of the domain expert are transformed into an implementation formalism. This *modeling* view of the knowledge acquisition process implies a need for tools that directly support both the domain expert and knowledge engineer in this effort.

The knowledge acquisition process does not culminate at some arbitrary point in the development life-cycle; but rather extends for the life of the system. Thus, the knowledge acquisition process encompasses the following facets (among others): elicitation and modeling of human expertise, testing the system's efficacy, refinement of the knowledge base, and maintenance of the resulting system. This *process* view implies that knowledge acquisition tools should support as many of these activities as possible.

---

[1]Inst. for Human & Machine Cognition, Div. of Comp. Science, University of W. Florida, Pensacola, FL 32514

[2]Cornell University

[3]Brock University

1.1 Automated Knowledge Acquisition Tools

Theories without tools are empty, tools without theories are blind.

— with apologies to Immanuel Kant

A commonly proposed partial solution to the difficulties associated with the knowledge acquisition process is the design and implementation of automated tools for interacting with domain experts, acquiring and organizing knowledge, and automatically generating a prototype expert system. To what extent are we justified in optimistic expectations about the usefulness of our tools? Upon what foundation may we base these sanguine anticipations? It seems clear that we may *expect* a proposed knowledge acquisition tool (as with any other tool) to be successful to the extent that we have a serviceable theory explaining the envisioned tool's basis of operation and delineating its approximate range of application. For example, we are able to satisfactorily judge the circumstances under which a thermometer is a useful tool only because we have a theory about its use.

However, as noted by Bradshaw and Boose (1989), a significant difficulty has been that much of this work lacks a plausible theoretical foundation.

> *As a consequence of incomplete theory and a limited repertoire of practical approaches to the dynamics of the modeling process, knowledge engineers have had to rely on intuition and experience as the primary means of developing and testing effective procedures.* (p. 6)

Rarely do working knowledge engineers explicitly consider the epistemological underpinnings of the methods and tools they employ in their task.

Many engaged in knowledge acquisition (as researchers or practitioners) may be classified as tool-makers and/or tool-users. Tool-makers should exploit theory as a means of building their tools on a sound footing and as a framework in which to make explicit their epistemological assumptions (Ford & Adams-Webber, 1990). Furthermore, theory may offer tool-makers a useful infrastructure upon which to build highly integrated collections of tools and techniques (i.e., hybrid knowledge acquisition workbenches). Tool-users may employ theory as the basis for a principled application of the myriad tools now becoming available. That is, a robust theory can endow a knowledge acquisition tool with the kind of conceptual supports (guides) that an operators manual alone could never provide.

Integration is the battle cry as tool-makers rush to produce hybrid knowledge acquisition tools. However, it is becoming widely realized[4] that *ad hoc* combinations of techniques and tools – a sort of Occam's hell -- may not contribute much to ameliorating the knowledge acquisition bottleneck. As we noted earlier, theory can offer an infrastructure upon which to build highly integrated hybrid knowledge acquisition tools in a principled way. ICONKAT is an effort in this direction.

The main sections of this paper are concerned with the following related topics: Section 2, Kelly's personal construct theory and its major methodological tool -- repertory grids; Section 3, Ausubel's assimilation theory and its major methodological

[4]General consensus expressed in the group meeting of the knowledge acquisition tools special interest group at the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff (cf. J. Bradshaw & B. Woodward, 1989)

tool -- concept maps; Section 4, ICONKAT, a constructivist hybrid knowledge acquisition tool based upon an integration of repertory grids and concept maps. The theoretical rational of this system is explicated in terms of principles from both personal construct theory and assimilation theory. Section 5 provides a general summary.

2. PERSONAL CONSTRUCT THEORY

Personal construct theory, as formulated by Kelly (1955, 1969, 1970) and summarized by Adams-Webber (1987), is essentially a constructivist model of human representational processes. The basic units of analysis in this theory are bipolar dimensions termed 'constructs'. Kelly posited that the underlying distinction which lends each construct meaning is dichotomous in form, for example, *happy/sad, young/old, odd/even*. He argued further that the dichotomous structure of personal constructs is an essential feature of the way in which people organize information (cf. Adams-Webber, 1990). For example, if the same event could be perceived simultaneously as equally pleasant and unpleasant in the same respect, then this distinction would have no definite meaning.

We use personal constructs to represent perceived similarities and differences among events; and then we organize these representations into coherent patterns or 'schemata' within the framework of which we are able to detect certain recurrent patterns in our experience over time and then feed these representations forward in the form of expectations about future events (cf. Ford, 1989). Each construct has a specific range of convenience, which comprises "all those things to which the user would find its application useful." Accordingly, the range of convenience of a construct defines its extension in terms of a single aspect of a limited domain of events (Mancuso & Eimer, 1982). A particular construct seldom, if ever, stands alone in our experience, because it is usually deployed together with one or more other hierarchically related constructs in interpreting and predicting events. Indeed, a necessary condition for organized thought is some degree of overlap between constructs in terms of their respective ranges of convenience (Adams-Webber, 1970). It is this overlap, or intersection, between the constructs' extensions that enables us to formulate hypotheses. That is, in interpreting an event we essentially categorize it in terms of one or more constructs, and then by considering related constructs, we can derive predictive inferences from that initial categorization.

2.1 Repertory Grids

Kelly's *Role Construct Repertory Grid* was developed to assess an individual's personal construct system, that is, "one's own finite system of cross references between the personal observations he has made and the personal constructs he has erected" (1965, p. 291). A repertory grid is essentially a complex sorting task in which a list of elements are judged successively on the basis of a set of bipolar constructs (Adams-Webber, 1987). These elements can be either concrete or abstract entities, and may be said to operationally define the grid's universe of discourse. They should be chosen carefully to completely represent the topic of interest, and also be roughly of the same type and level of complexity.

Constructs represent the ways in which elements are judged to be similar or different from each other: that is, they permit the subject to make relevant distinctions among the elements. Thus, a repertory grid may be considered a mapping of elements onto constructs. The data generated by each subject are entered into a separate two-dimensional table, or 'grid', in which there is a column for every element and a row for every construct. Each row-column intersect in this table contains a number indicating how a given construct was applied to a particular element.

The repertory grid in Figure 1 was produced, using ICONKAT, by a domain expert (radiologist) participating in the development of a large-scale expert system for the diagnosis of heart wall motion abnormalities.

Rate elements on a scale from 1 to 3; a '1' is at the LHP, a '3' is at the RHP			ELEMENTS 1 2 3 4 5 6 7 8
C1	central crescent	dispiced or absent	1 1 2 3 3 2 2 3
C2	no blue fingers	blue finger	1 2 2 3 3 1 1 3
C3	concentric	asynchronous	1 2 2 3 3 1 1 3
C4	intact	interrupted	1 1 2 2 2 1 1 3
C5	improve/exer	worsen/exer	1 2 2 3 2 2 2 3
C6	symmetric	assymetric	1 1 2 3 3 1 2 3

Element Index:
1 - Normal
2 - Nonspecific Wall Abnormalities
3 - Nonspecific WMA/MVP
4 - Suspicion for Ischemia
5 - Abnormal Fx/Ischemia
6 - Abnormal Fx/Cardiomyopathy
7 - Abnormal Fx/Valve Disease
8 - Severe LV Dysfunction

Figure 1: A repertory grid from domain of nuclear cardiology

During the last thirty years there has been a considerable proliferation of new forms of repertory grid in which people, objects, situations, or other kinds of elements, are either categorized, rated, or rank-ordered on a set of constructs. The act of the respondent's assigning a rating to an element on a given construct has been interpreted in a variety of ways, which has led to several different approaches to grid analysis, including information theoretic measures, non-parametric factor analysis, conventional factor analysis, principal component analysis, multidimensional scaling, and hierarchical cluster analysis, among others (Adams-Webber, 1987). These procedures have been widely used in clinical psychology, as well as in the study of education and management (cf. Adams-Webber & Mancuso, 1983). In addition, several recent attempts to aid the knowledge acquisition process have been based either loosely or directly on personal construct psychology. Specifically, repertory grid analysis is becoming an increasingly useful tool in knowledge acquisition for expert systems (Boose, 1986; Gaines & Shaw, 1980, 1986; Ford, Petry, Adams-Webber, & Chang, 1990).

Nonetheless, the repertory grid is not a 'magic matrix': the root of the method's relative success lies in the fact that the repertory grid is the principal methodological component of personal construct theory -- an epistemologically sophisticated psychology (Ford & Adams-Webber, 1990). Within the framework of personal construct theory, the judicious use of repertory grids may assist the knowledge engineer in avoiding the domain expert's cognitive defenses, and thus elicit deep knowledge that the expert would not have been able to express otherwise.

3. ASSIMILATION THEORY

Ausubel's (1963, 1978) assimilation theory, as does Kelly's (1955) personal construct theory, belongs to the family of theories contributing to a constructivist model of human representational processes. Assimilation theory is essentially a cognitive theory of learning and education. Ausubel argues that learning is synonymous with a change in the meaning of one's experiences. His fundamental premise seems deceptively simple:

> *Meaningful learning results when new information is acquired by deliberate effort on the part of the learner to link the new information with relevant, preexisting concepts or propositions in the learner's own cognitive structure.* (Ausubel, 1978; p. 159)

Ausubel's theory posits that meaningful learning involves the assimilation of new concepts and propositions into existing cognitive structures. In this context, concepts are defined as perceived regularities in events or objects (or the records of events or objects) which have been designated by a label. This assimilation of new meaning leads to a progressive differentiation of and reintegration cognitive structure.

Ausubel's theory deals with the processes involved in linking information to existing segments of cognitive structure in a propositional manner. He explicates various forms of meaningful, as opposed to rote, learning that involve the assimilation of new information. Ausubel assumes that meaningful learning requires that the learner's cognitive framework contain relevant anchoring ideas to which the new material can be related. Indeed, he argues that "the most important single factor influencing learning is what the learner already knows. Ascertain this and teach him accordingly." Specifically, 'concept maps' developed by Novak (1977), have been designed to tap into a person's cognitive structure and externalize one's concepts and propositions. In the next section, we will describe concept mapping, assimilation theory's major methodological tool for ascertaining what is already known.

3.1 Concept Maps

A concept map is a two-dimensional representation of a set of concepts constructed so that the interrelationships among them are evident. The vertical axis expresses a hierarchical framework for the concepts. More general, inclusive concepts are found at the highest levels, with progressively more specific, less inclusive concepts arranged below them. Maps emphasize the most general concepts by linking them to supporting ideas with propositions.

Along the vertical axis, maps display Ausubel's concept of subsumption, namely that new information often is relatable to and subsumable under more inclusive concepts. Along the horizontal axis, maps express the idea of progressive differentiation by showing how concepts acquire richer meaning as new propositions are acquired. This axis shows that meaning making is a continuous process. Concepts are always being modified and made more explicit and inclusive.

The overall structure of a concept map creates a hierarchical framework for the concepts included in it. All concepts at a given level in the hierarchy will tend to have a similar degree of generality. Consider the concept map about concept mapping in Figure 2 (by convention links run top-down unless annotated with an arrowhead).

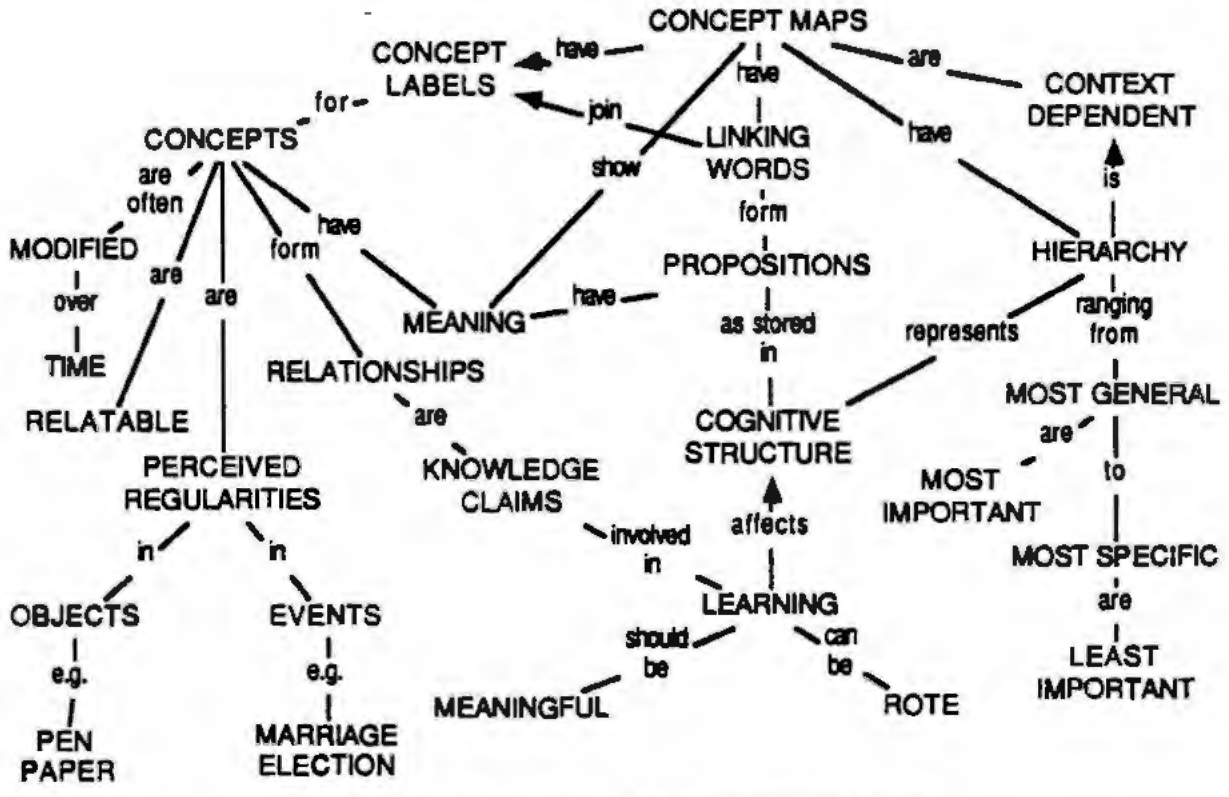


Figure 2: A concept map about concept mapping

Concept maps represent meaningful relationships between concepts in the form of propositions. Propositions are two or more concepts linked by words to form a semantic unit. In the simplest form, a concept map would be just two concepts connected by a linking word to form a single proposition. For example, "grass is green" would represent a simple map forming a valid proposition about the concepts "grass" and "green". A concept acquires additional meaning as more propositions begin to include the concept. Thus, "grass is green", "grass is a plant", "grass grows" and so on, all expand the meaning of the concept "grass". In this sense, concept maps represent concept meaning in a framework of embedded propositions.

In an educational setting, concept mapping techniques have aided people of every age to examine many fields of knowledge (Novak & Gowin, 1984). This rich expressive power derives from a map's ability to allow its creator the use of a virtually unlimited set of linking words to show how meanings have been developed. When concepts and linking words are carefully chosen, concept maps are powerful tools for observing nuances of meaning. Mapping techniques have also been employed to help students "learn how to learn" by bringing to the surface one's cognitive structure and self-constructed knowledge. Problems of exactly this sort currently face the knowledge acquisition community. As a knowledge elicitation technique, concept maps provide a framework for eliciting the most significant part of a specialist's expertise, that is, his/her personally constructed knowledge (Ford & Adams-Webber, 1990).

The structure of a concept map is dependent on the context of the map's application. Consequently, maps having similar concepts can vary from one context to another and are highly idiosyncratic. The strength of concept maps lie in their ability to measure a particular person's knowledge structure about a given topic in a specific context (Novak & Gowin, 1984). There is evidence that when experts and novices in the same domain construct concept maps, the former produce maps which are both more differentiated and integrated (Novak, 1989).

4. ICONKAT: Integrated Constructivist Knowledge Acquisition Tool

The constructivist tool described here (i.e., ICONKAT) interactively assists the domain expert in the task of building a model of his/her expertise. In keeping with the view that the knowledge acquisition process continues for the life of the system, ICONKAT provides explicit support for the elicitation and modelling of human expertise, documentation, testing, and maintenance of the knowledge base under development.

A current trend in applied AI (e.g., expert system design) is the development of integrated, embeddable performance environments (e.g., NEXPERT OBJECT™) which offer knowledge engineers both object-oriented knowledge bases and rule-structured inferencing mechanisms. However, this development now requires that automated knowledge acquisition tools generate much more than mere production rules when creating prototype systems. ICONKAT interactively supports the domain expert's elaboration of the relevant object space by use of concept maps, as well as generating the associated rules from tightly-coupled repertory grid data. An overview of the entire system is presented in Figure 3.

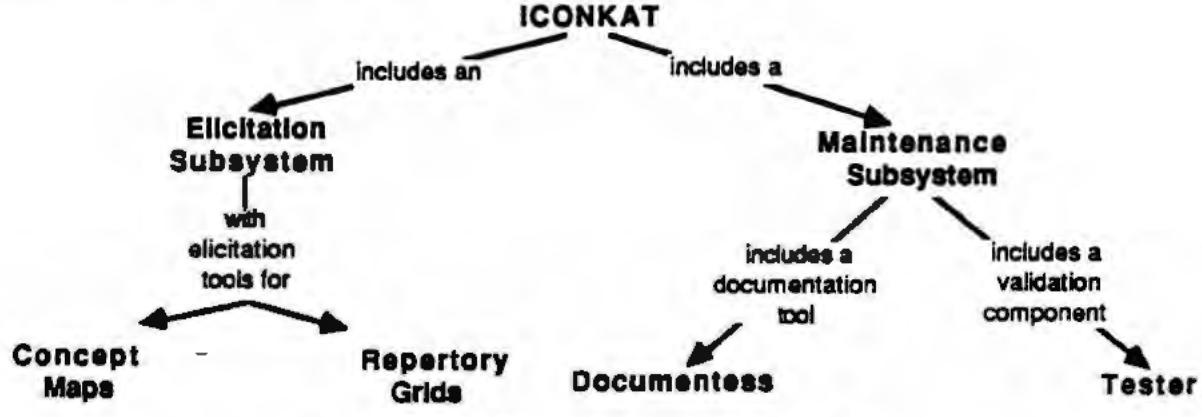


Figure 3: ICONKAT System Architecture

4.1 The Elicitation Subsystem

ICONKAT interacts with an expert through a concept mapping and repertory grid interface (see figure 4). The system uses concept maps to elicit the object space from the domain expert, incrementally generating a hierarchically-structured knowledge base as a map is created. The knowledge base created from the map is provided to the representation dimension of the performance environment. Concepts in the map are specified by the user as either classes, objects, subobjects or properties. The links are used to propagate inheritance appropriately.

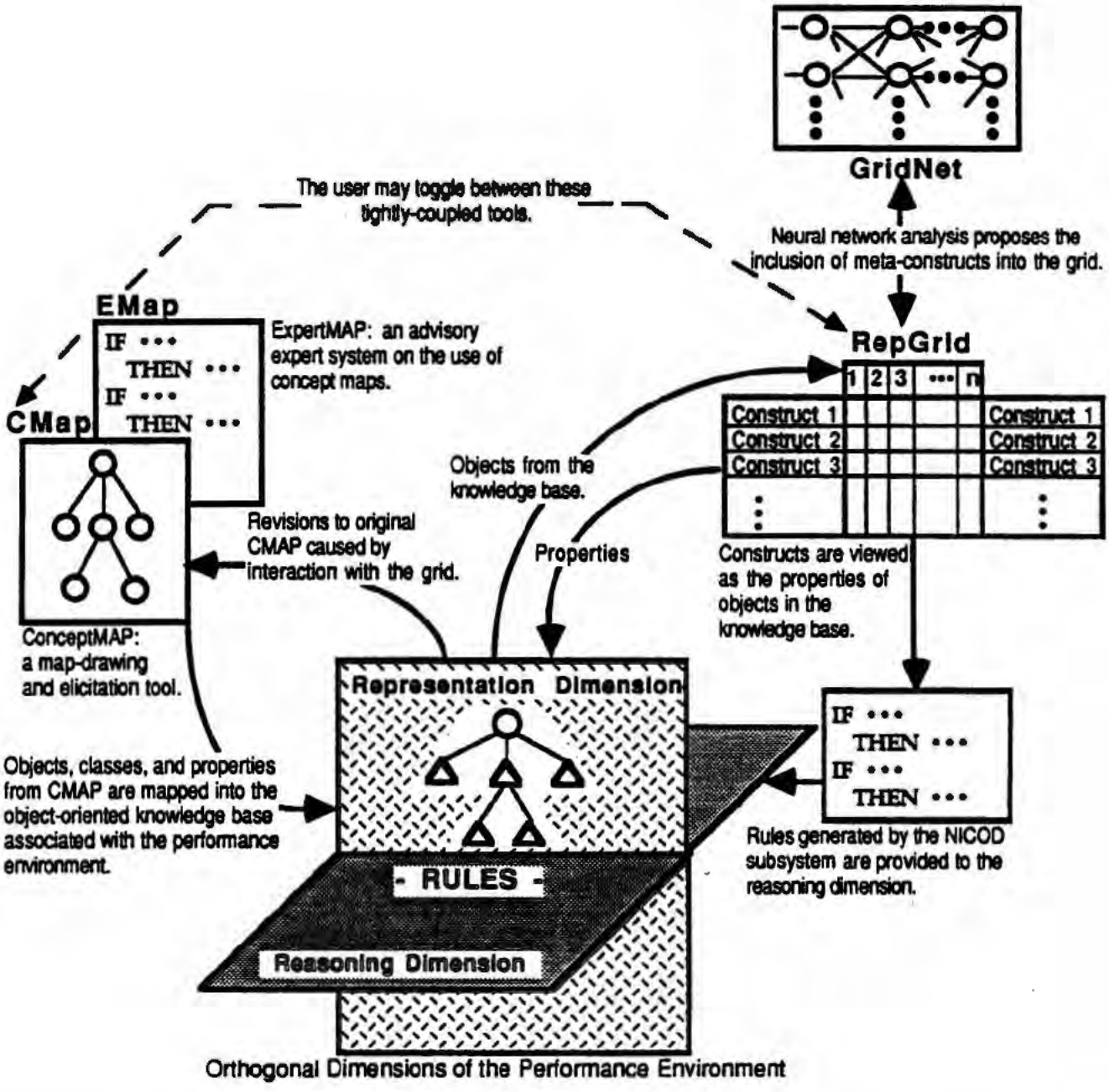


Figure 4: Elicitation Subsystem Overview

ICONKAT elicits the reasoning dimension by the use of repertory grids. The hierarchically-grouped concepts found in the elicited concept map provide the elements for a single repertory grid. A hierarchical collection of grids can, if necessary, be elicited on the basis of the object space, with each grid at a different level of abstraction. Using GridNet (the neural net construct elicitation method described in section 4.1.2.1), as well as triads and other well-known repertory grid techniques (cf. Adams-Webber, 1979), constructs are elicited on the basis of the elements already defined as objects in the knowledge base. The constructs can then serve as the properties of these objects. Further analysis of the repertory grids will determine the construct entailments necessary to generate production rules. Thus, the system creates a reasoning space structured on the basis of the object space already represented.

4.1.1 The Concept Mapping Components

The concept mapping elements of ICONKAT are primarily based on CMap (Hunter, Stahl, & Novak, 1990), a mapping elicitation tool for assisting in the design and construction of concept maps. CMap provides a graphical concept mapping environment which users can employ to codify their knowledge of a specific domain. Initially, the user supplies concepts to the drawing program one at a time, and later as the user develops the map, propositions can be included by linking concepts together. Propositions can, if desired, be left unlabelled to signify a relationship between two concepts which, for whatever reason, cannot be explicitly named at the time the link is established.

To understand the powerful visual component of CMap, consider the screen shot of the system in operation show in Figure 5. The sample screen displays a concept map (reproduced from Novak & Gowin, 1984) as it appears in the CMap environment.

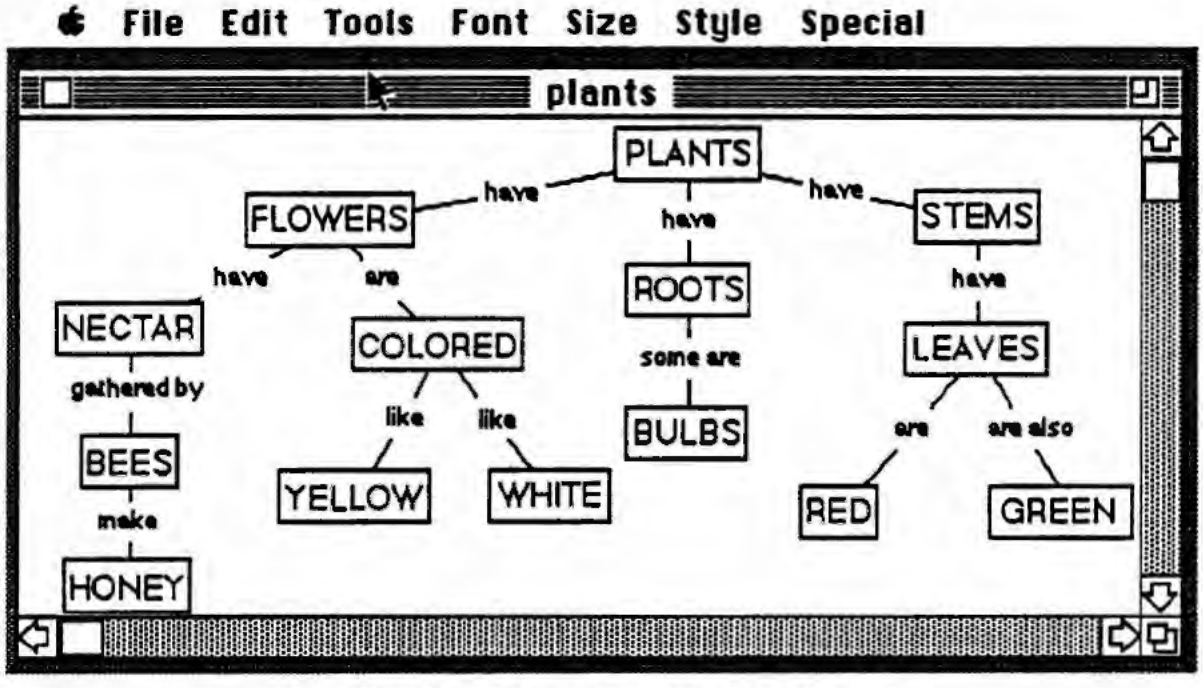


Figure 5: A Sample Screen From the CMap Subsystem

During an interactive session with CMap, users are free to edit, remove or relocate the concepts and links in their map. Throughout this entire development process, maps are continuously displayed in their two-dimensional graphical representation. This constant visual feedback provides essential information to users when evaluating the effectiveness of a specific modification or alteration, and empowers them to easily consider alternative conceptual hierarchies, thereby constructing the most effective map, the one which best conveys the desired meaning and relationships they are working to express.

The concept mapping portion of ICONKAT also includes a real-time advisory expert system (known as EMap) which assists the domain expert in creating well structured maps. EMap responds to user-initiated requests for advice about the map under

construction with information about actions the user might reasonably take to develop and improve his/her map. By monitoring the structure of the concept map under development, EMap can focus one's attention on regions within a concept map in need of restructuring or further elaboration. EMap aids the process of knowledge base construction, by exploiting both the tool's underlying theory (assimilation theory) and the heuristic knowledge of its inventor -- Joe Novak served as the domain expert.

In application to the nuclear cardiology domain, we have found this concept space elicitation method very successful. The concept map in Figure 6 was produced, using ICONKAT, by a domain expert after assimilating some of the map-structuring advice suggested by EMap.

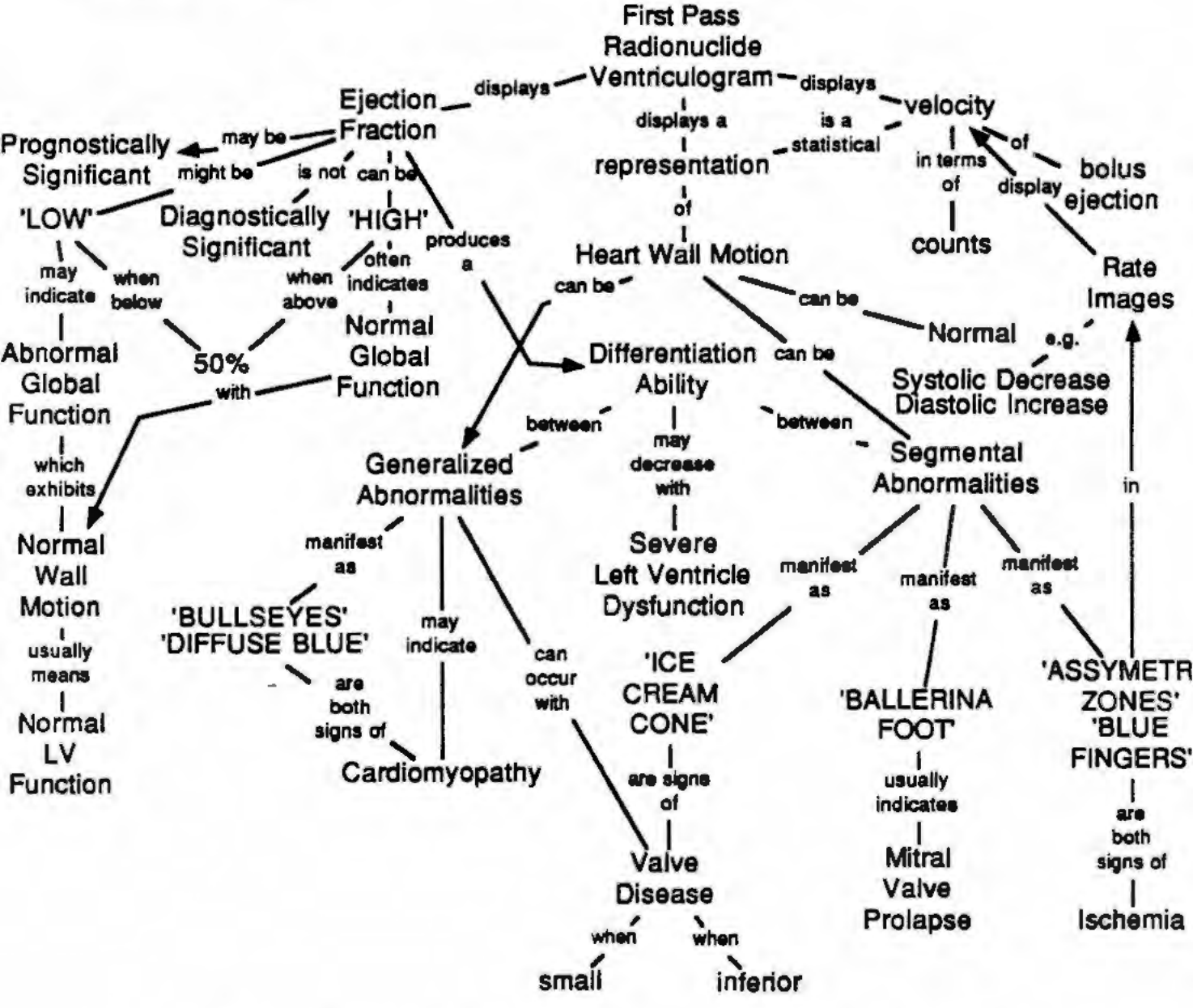


Figure 6: A Segment of a Concept Map From the Domain of Nuclear Cardiology

The concept map provides a high-level conceptual overview of the domain as constructed by the expert. In particular, this map expresses relationships among ejection fraction (a critical numerical value), other manifestations of heart wall image abnormalities (e.g., 'Blue Fingers'), specific heart diseases (e.g., ischemia) and human physiology. Moreover, the relevant disease states upon which this expert renders a diagnosis can be found at the lowest levels of the map and have been incorporated as elements in the repertory grid components of ICONKAT. Note that the map includes the domain expert's personally constructed expertise in the form of visual analogies that he employs as markers for perceived image abnormalities (e.g., 'Blue Fingers', 'Blue Bullseyes' and 'Ice Cream Cones'). These markers are the basis on which the expert differentiates the various disease states, and as such, may be included as constructs in the developing repertory grid.

In summary, the concept mapping components of ICONKAT assist the user to concentrate on introspection, and facilitate the elicitation of personally constructed knowledge, which in our view, represents the most significant and elusive portion of one's expertise (Ford & Adams-Webber, 1990).

4.1.2 The Repertory Grid Components

The repertory grid components of ICONKAT are largely based on NICOD (Ford et al., 1990), OMNIGRID (Mitterer & Adams-Webber, 1988), and the new neural net method for construct elicitation described in Section 4.1.2.1 (Jones & Ford, 1990). The repertory grid components of ICONKAT are illustrated in Figure 7.

NICOD is a semi-automated knowledge acquisition system supporting the elicitation, representation, and analysis of repertory grid data. It has been used by radiologists to quickly produce useful rule-sets without undue difficulty in several medical domains, including mammography and nuclear cardiology. NICOD is grounded on a theory of confirmation that incorporates the basic tenets of personal construct theory directly into the logic as a basis for the determination of relevance, thus strengthening the logic, and extending personal construct theory. More specifically, Kelly's notion of *range of convenience* is used to operationally define the range of relevance of hypotheses which are derived from relations of entailment between constructs. The concept of *a-planes* (Ford, 1987) is used for the binary decomposition of repertory grid data elicited from a domain expert. This procedure furnishes the uniquely determined strings of incidences required for the application of Bundy's (1985) truth functional incidence calculus.

OMNIGRID is a general tool for designing, administering and analyzing repertory grids that affords a variety of different formats, including rating scales, nominal categorization and rank-order procedures, with either elicited or supplied constructs and elements. In the analysis of repertory grid data elicited by this program, we are able to select from a wide range of composite indices and summary statistics tailored to the specific format options: these include, construct intercorrelations, element distances, information statistics and rating extremities.

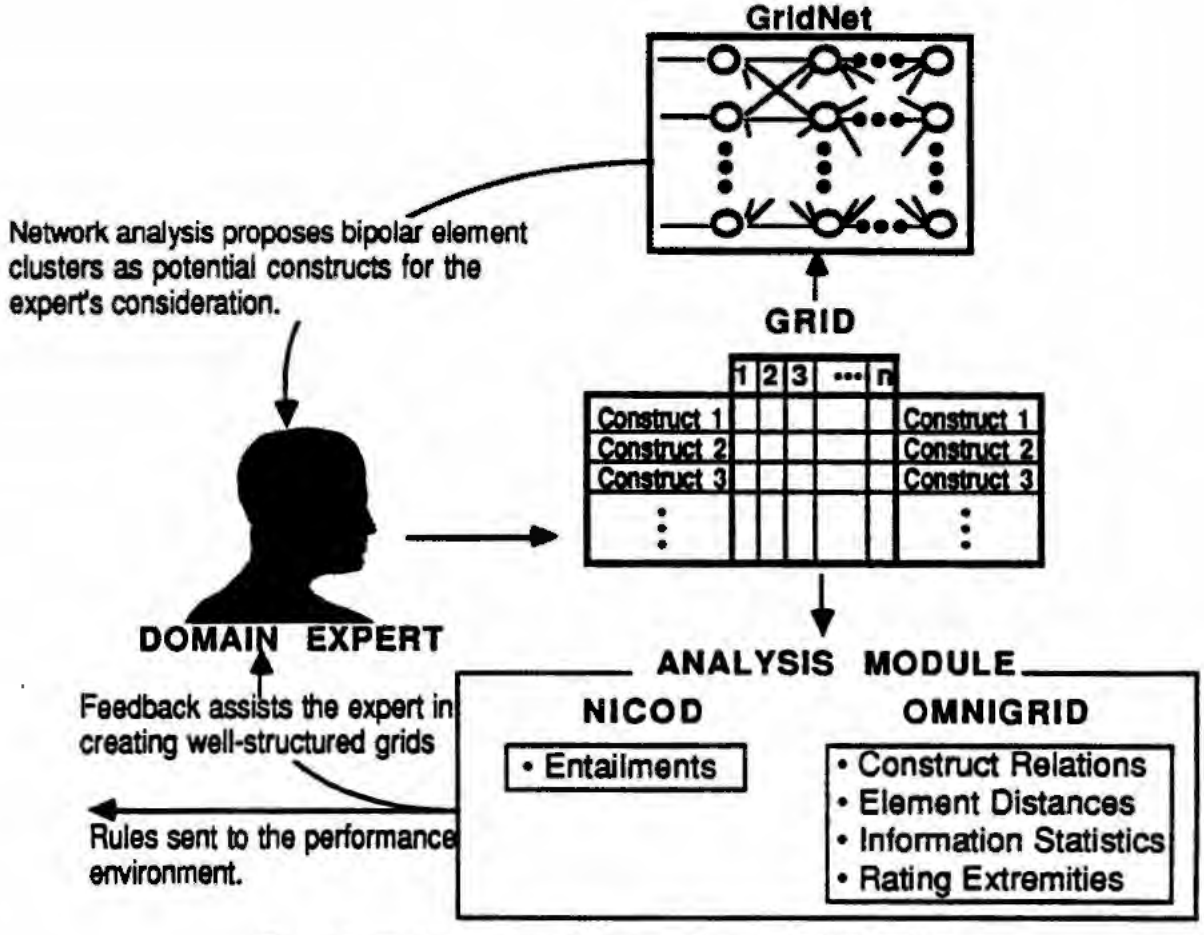


Figure 7: The Repertory Grid Components of ICONKAT

4.1.2.1 GridNet: A Neural Net Approach to Construct Elicitation

Although the repertory grid method of knowledge acquisition has proved quite successful, there remains room for considerable improvement. One difficulty with repertory grid oriented knowledge acquisition tools is the domain expert's inability to provide a reasonably complete sample of their constructs. This problem may stem partly from the limitations of currently available methods of construct elicitation. It is somewhat ironic that a knowledge acquisition tool should suffer from this problem.

Typically, constructs are elicited by asking the expert to compare the grid's elements (which are usually listed first) with each other in groups of two or three. When groups of two are used, the expert is asked to name a construct that distinguishes between them. When Kelly's (1955) triad method is used, the expert is asked to name a characteristic which two elements have in common which distinguishes them from the third. As noted by Butler and Corter (1986), there are logical problems for construct elicitation methods that rely exclusively on questions about grouped elements. Contrasting two elements with a third can fail to discover an important construct (characteristic) that distinguishes the pair. Conversely, if two objects are contrasted with one another, an important common characteristic may be

overlooked. These objections can be met if the knowledge engineer asks the expert to (1) give the common characteristics of a pair of elements A and B, and (2) give the features that A has that B does not (distinctive characteristics of A) and that B has that A does not (distinctive characteristics of B) (cf. Tversky, 1977). However, this technique suffers from the problem that the number of pairs and triples increases exponentially with the number of elements.

GridNet, the neural net element of ICONKAT, assists the expert to uncover non-linear patterns, as they occur, during the construction of a repertory grid, suggesting higher level 'meta-constructs' (see Figure 7). Kelly (1955, 1970) implies that relatively superordinate constructs might function as pivotal axes of reference in terms of which alternative construct subsystems can be logically integrated at higher levels of abstraction.

The expert's preliminary repertory grid data will serve as input to a self-organizing, multilevel, artificial neural net which uses back propagation, in which abstractions taken from this non-linear hierarchy, called 'hidden features', can be identified. These artificial neural net abstractions are then fed back as suggestive prompts to the domain expert for the elicitation of new constructs. Artificial neural net learning algorithms seem especially well suited to the extraction of meaningful patterns from repertory grid data that can then be used as useful prompts for the elicitation of new constructs. We anticipate that some of the constructs which we elicit by this method will prove to be relatively superordinate.

Latent features (i.e., bipolar element groupings) are derived from repertory grid data in the following manner. The construct vector patterns (i.e., rows; see Figure 1) from a given repertory grid are repeatedly and randomly input to a basic feedforward network during the back propagation net training session. In each case the data are compressed through only four initial hidden units, and then are expanded to produce an output as close to the original input vector as possible.
To the extent that this identity mapping succeeds for all six vectors, the first layer of hidden units can be said to have captured or encoded the structure of the entire repertory grid in the following respect: the output from each hidden unit is the threshholded inner product of the input pattern vector with a weight (feature) vector describing the hidden unit. This scalar product determines the component or amount of the feature vector contained in an input pattern in the ordinary Euclidean Hilbert Space sense.

A feature embodies one of the most recurrent patterns in the expert's point of view as represented by the multilayer, non-linear structure, extracted from the repertory grid by the network learning algorithm. Note that since multilayer connectionist systems are generally highly non-linear and hierarchical, these recurrent features may be latent within a number of hierarchical views, where each feature is the input cluster to a deeper hierarchy of clusters. It is this capacity for uncovering many layered, non-linear hierarchies that enables the elicitation of new constructs for possible subsequent use in the expert's expanded repertory grid.

GridNet and the knowledge engineer assist the domain expert in completion of a repertory grid as follows:

- first the domain expert provides the elements (perhaps with the assistance of the concept mapping components of ICONKAT), and then the knowledge engineer elicits an initial collection of constructs on the basis of these elements using standard techniques (e.g., triads);
- the expert rates the elements on the constructs available thus far -- at this point the knowledge engineer may ask GridNet (running in the background) to provide a feature set that replicates the expert's current grid;
- each of the qualitative features (in a given feature set) are presented to the expert as potential bipolar groupings in which the elements are clustered at the opposing poles, and the expert is asked to provide a construct for each grouping;
- the domain expert now returns to the repertory grid and rates the elements on the basis of the newly derived constructs;
- the knowledge engineer may at any time invoke ICONKAT's analysis module (i.e., NICOD and OMNIGRID) to produce a rule-set or other analysis of the newly expanded repertory grid.

The feature set of bipolar element clusters illustrated in Figure 8 are derived from the repertory grid in Figure 1.

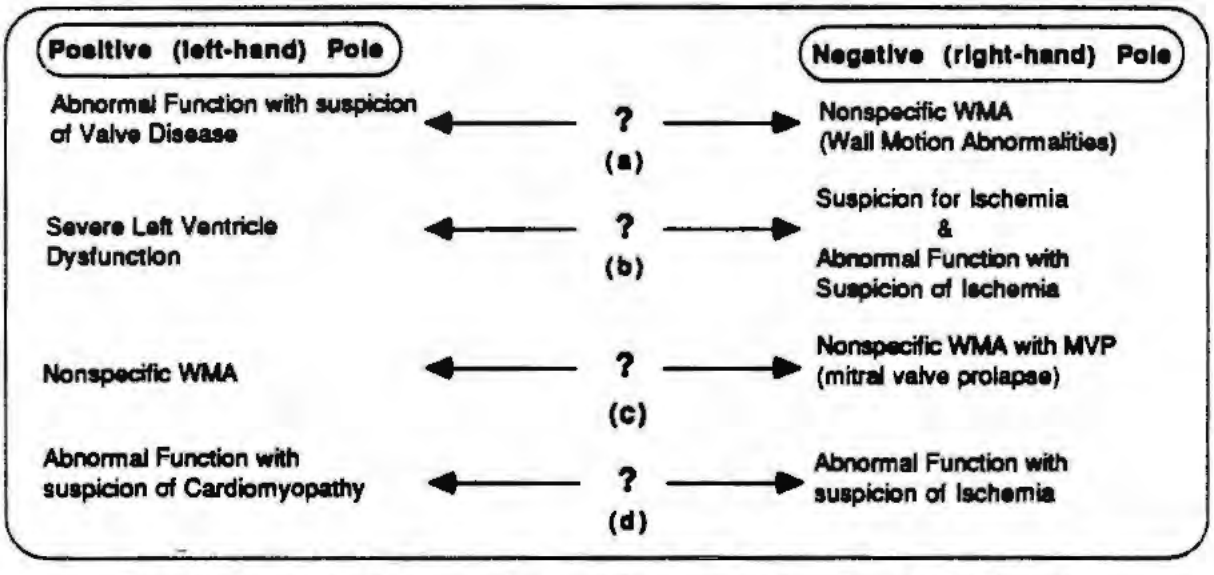


Figure 8: Extracted grid features (potential constructs) from the grid in Figure 1

The domain expert is asked to provide a bipolar construct to differentiate the element clusters from each other. Inasmuch as this clustering maximally separates the elements in some decision space, the proposed features may tend to be more useful in teasing out missing constructs (as reflected in an overall structure in the repertory grid) than features arising from less distinct sets of elements.

For example, consider the first feature vector as presented to the domain expert in Figure 8(a). GridNet asks the expert to furnish a bipolar construct which distinguishes the disease state, *Abnormal Function with suspicion of Valve Disease*, from the

disease state, *Nonspecific Wall Motion Abnormalities*. In other words, "How is the first group of diseases (in this case only one) most significantly different from the second group." If the expert cannot provide a construct for this feature, the next item in the feature set will be presented to the expert in a similar fashion. This process continues until the domain expert has provided a construct corresponding to one or more of the features presented.

In application to the nuclear cardiology domain, we have found this construct elicitation method very successful. For example, the domain expert was able to provide a construct for both feature (b) and feature (c) from Figure 8. The construct represented by feature (b),*segmental/generalized*, is a meta-construct (i.e., a higher-level, pivotal construct). Note the important position the domain expert accorded this new construct when he incorporated it into the concept map in Figure 6. The construct represented by feature (c), *no ballerina foot/ballerina foot*, is the fundamental distinction used by the domain expert to discriminate Mitral Valve Prolapse from other heart wall motion abnormalities. In summary, the GridNet approach to construct elicitation assisted the expert to expand the repertory grid in a meaningful way, thus producing an improved expert system.

4.1.3 Integration of the Elicitation Subsystem: Integrated How and Why

The components of ICONKAT's elicitation subsystem are integrated on three levels, which we will proceed to briefly examine in a top-down fashion. In general, our efforts are directed toward the development of a unified elicitation subsystem informed by a constructivist approach to knowledge acquisition.

First, and most profoundly, the principal components of ICONKAT's elicitation subsystem are integrated at the *theoretical* level. We (as tool-makers) have exploited the considerable epistemological overlap between Kelly's (1955) personal construct psychology and Ausubel's assimilation theory (Ausubel, Novak & Hanesian, 1978). It is this theoretical overlap that enables each theory's major methodological tool (i.e., repertory grids and concept maps) to be integrated in a principled manner. Furthermore, this common theoretical framework, contributing to a constructivist model of human representational processes, lends ICONKAT the kind of conceptual supports (guides) that a user's manual alone could never provide.

Second, in the ICONKAT system, repertory grids and concept maps are integrated at the level of *practice* (domain level). The similarities of personal construct psychology and assimilation theory, as well as an appreciation of the differences in their intended ranges of application (i.e., psychology and education, respectively), empower the knowledge engineer to employ these otherwise orthogonal tools in a symbiotic fashion. Specifically, concept maps provide 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 which inform the knowledge engineer about the potential and appropriate use of grids within this domain. For example, the hierarchically-grouped concepts found in the elicited concept map may provide the elements for a single repertory grid, or a collection of grids, based on the hierarchical structure found in the map.

Third, the components of ICONKAT's elicitation subsystem (i.e., CMap and RepGrid) are integrated at the *representation* level. CMap and RepGrid utilize the representation facilities of an embedded performance environment (e.g., NEXPERT OBJECT™) as a mediating representation (as described previously in section 4.1).

4.2 The Maintenance Subsystem

Consistent with the view of knowledge acquisition as an ongoing process (rather than a discrete phase), ICONKAT includes a maintenance subsystem. This subsystem provides support tools for use by the knowledge engineering team when testing the system's performance, refining the knowledge base, and maintaining the overall system. The maintenance subsystem of ICONKAT includes two hypermedia components which fully exploit the integrated, embeddable features of some of the more recent hybrid AI performance environments (e.g., NEXPERT OBJECT™).

Hypermedia systems are an approach to information management in which data is stored in a network of nodes connected by links. Nodes can contain text, graphics, audio, video as well as other forms of data. The entire system itself is meant to be viewed through an interactive browser allowing the user to search for topics of interest and then follow links to various cross references and other relevant entries. Unlike conventional documentation which is meant to be read from beginning to end following the same sequence, the path of the reader in a hypertext system becomes a network that can crisscross entire documents. Overall, hypermedia systems have begun to produce a qualitative change in the way people conceptualize information resources (e.g., knowledge bases) since the organizational and cross-reference structures of the text links and nodes can be changed dynamically.

4.2.1 Documentess (Documentation Tool for Expert Systems)

Documentess is an interactive 'intelligent' documentation tool for supporting and maintaining expert system knowledge bases. Exploiting the flexibility offered by hypermedia in structuring information, Documentess provides a single integrated environment for all the various informational components of the knowledge acquisition process, including, documentation, maintenance, and also support for the subsequent expert system's explanation facility.

Essentially, Documentess provides a mechanism for hierarchically structuring a relatively unstructured knowledge base. Typically, knowledge bases are not organized from the perspective of humans, but rather for the convenience of the representation and reasoning mechanisms of the performance environment. Documentess 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. Documentess directly inherits its supporting hierarchy from the model of the domain as constructed by the knowledge engineer on the basis of the expert's previous interactions with the elicitation subsystem. The previously elicited concept maps and repertory grids can provide the knowledge landscape in which the user may browse.

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 performance environment.

In summary, an appropriate hypermedia system integrated with an embedded performance environment enables the knowledge engineer to move smoothly between the methodological tools employed in the knowledge acquisition process and the resulting rules and objects which comprise the knowledge base of the expert system under development. In addition to its role as a repository for the domain model elaborated during the elicitation process, Documentess makes possible the development of an explanation subsystem of the kind described below.

Explanation as a knowledge acquisition issue

A knowledge-based system typically evolves over its lifetime in a largely *ad hoc* fashion, thus potentially endangering the integrity of the carefully constructed domain model. This approach often leads to a 'runaway' expert system, in which the actual knowledge base slides ever further out of conformance with the documentation and explanation modules, thus making the performance of its explanation facility unsatisfactory to users, and effectively rendering the knowledge base unmaintainable.

A 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 representations formed during the knowledge acquisition process. Without these mediating representations, the implicit connections that establish the 'logical' structure of the domain model are lost, and as a result, much effort will be required to 'put Humpty Dumpty back together again'. This is precisely the task (i.e., reassembling Humpty) facing knowledge engineers who do not regard explanation as part of the knowledge acquisition process and, lacking an adequate model, attempt to build their explanation systems *post hoc*.

Thus, in addition to its role as a documentation/maintenance tool, Documentess offers a flexible embodiment of the domain model for subsequent exploitation by the explanation subsystem. The capacity of most current expert systems to explain their findings (i.e., conclusions), tend toward inadequate causal descriptions of the behavior of the performance environment's reasoning mechanism. Because Documentess affords a domain model view of the knowledge base, explanations can be provided at a variety of conceptual levels. Upon a user-initiated request for explanation, the context-sensitive explanation subsystem (executing in an integrated hypermedia environment) will empower the user to explore the domain model.

4.2.2 Tester

The Tester, as its name implies, is a hypermedia system for evaluating the performance of knowledge-based systems. The tool volunteers data to the performance environment, extracting values as needed from either a database or a collection of spreadsheet files. By monitoring and recording the resulting inference process, specific components of an expert system can be evaluated. This tool helps partially automate the validation phase, a crucial step in the development of an expert system.

5. SUMMARY

The main focus of this paper has been the results of our research aimed at the development of an integrated constructivist knowledge acquisition system, ICONKAT. As noted at the outset, the most fundamental step in the knowledge acquisition phase of the construction of an expert system is the elicitation of knowledge from a skilled individual. We have suggested elsewhere (Ford & Adams-Webber, 1990), that human 'expertise' not only consists of explicit knowledge of the sort found in textbooks (i.e., widely shared consensual beliefs); but also includes a fund of personal experience comprising functional, but fallible anticipations held with high confidence and uncertain validity (cf. Agnew & Brown, 1989). We have argued also that *ad hoc* combinations of techniques and tools cannot contribute significantly to resolving most of the difficulties associated with the knowledge acquisition process.

Specifically, we examined some of the epistemological assumptions of two related constructivist models of cognition. First, we explicated some of the central principles of Kelly's personal construct theory and its major methodological tool, repertory grids. Secondly, we summarized Ausubel's assimilation theory and described its major methodological tool, concept mapping.

Finally, we introduced a new constructivist knowledge acquisition tool (ICONKAT) supporting the entire knowledge acquisition process, which in our view, encompasses elicitation, model construction, representation (i.e., implementation), documentation, testing, and maintenance. The ICONKAT system architecture consists of two major subsystems; elicitation and maintenance.

The elicitation subsystem integrates repertory grids and concept maps. Its theoretical rationale was explicated in terms of some basic tenets of personal construct theory and assimilation theory. In addition, we discuss a new method of construct elicitation exploiting neural nets to elicit new constructs on the basis of previously elicited repertory grid data. Integrated in a principled manner, the resulting human-machine system aims to bring the expert's self-constructed knowledge to the surface.

The maintenance subsystem of ICONKAT provides support tools for use by the knowledge engineering team when testing the system's performance, refining the knowledge base, and maintaining the overall system. In particular, the maintenance subsystem includes two hypermedia components. Documentess provides a single

integrated environment for all the various informational components of the knowledge acquisition process, including, documentation, maintenance, and also support for the subsequent expert system's explanation facility. Tester is a hypermedia system for evaluating the performance of knowledge-based systems, thus partially automating the validation phase.

ICONKAT is currently employed in the design and construction of an expert system for the diagnosis of first pass cardiac functional images, a non-invasive radionuclide technique used to evaluate heart wall motion abnormalities.

We anticipate that personal construct theory, assimilation theory and other constructivist models of cognition will continue to provide useful conceptual supports (guides) for the development of practical tools for knowledge acquisition.

6. ACKNOWLEDGMENTS

Many people have influenced the development of this work. We owe particular thanks to Jim Andrews, John Brennan, John Coffey, Jack Fleming, Scott Hunter, Johnny Liseth, Steve Venticique, and Jeff Yerkes. This work was supported in part by Florida High Technology Enhancement Funds.

7. REFERENCES

Adams-Webber, J. R. (1970). Actual structure and potential chaos. In D. Bannister (Ed.), *Perspectives in Personal Construct Theory* (pp. 30-45). London: Academic Press.

Adams-Webber, J. R. (1979). *Personal Construct Theory: Concepts and Applications*. New York: Wiley.

Adams-Webber, J. R. (1987). Personal construct theory. In R. Corsini (Ed.), *Concise Encyclopedia of Psychology* (pp. 824-825). New York: Wiley Interscience.

Adams-Webber, J. R. (1989). Kelly's pragmatic constructivism. *Canadian Psychology*, 30, 190-193.

Adams-Webber, J. R. (1990). Some fundamental asymmetries in the structure of personal constructs. In G. Neimeyer & R. A. Neimeyer (Eds.), *Advances in Personal Construct Psychology* (49-85). Greenwich, CT: JAI Press.

Adams-Webber, J. R. & Mancuso, J. C. (1983). The pragmatic logic of personal construct psychology. In J. R. Adams-Webber & J. C. Mancuso (Eds.), *Applications of Personal Construct Psychology*. New York: Academic Press, pp. 1-10.

Agnew, N. M. & Brown, J. L. (1989). Foundations for a theory of knowing: II. Fallible but functional knowledge. *Canadian Psychology*, 30, 168-183.

Ausubel, D. P. (1963). *The Psychology of Meaningful Verbal Learning*. New York: Grune and Stratton.

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

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

Bradshaw, J. M. & Boose, J. H. (1989). Decision analytic techniques for knowledge acquisition: Combining situation and preference models using AQUINAS. *International Journal of Man-Machine Studies*.

Bradshaw, J. M. & Woodward, B. (1989). Report of the working group on knowledge acquisition tools. In J. Boose and B. Gaines (Eds.), *Knowledge Acquisition for Knowledge-Based Systems Newsletter*, vol. 1, no. 3, pp. 12 - 13.

Bundy, A. (1985). Incidence calculus: A mechanism for probabilistic reasoning. *J. of Automated Reasoning*. vol. 1, pp. 263-283.

Butler, K. A. & Corter, J. E. (1986). Use of psychometric tools for knowledge acquisition: A case study. In W. A. Gale (Ed.), *Artificial Intelligence and Statistics*. Reading, MA: Addison-Wesley, pp. 295-319.

Fischler, M. A. & Fischein, O. (1987). *Intelligence: The Eye, the Brain, the Computer*. Reading, MA: 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. M. (1989). A constructivist view of the frame problem in artificial intelligence. *Canadian Psychology*, 30, 188-190.

Ford, K. M. & Adams-Webber, J. R. (1990). Knowledge acquisition and constructivist epistemology. In R. Hoffman (Ed.), *The Cognitive Psychology of Expertise: Empirical Approaches to Knowledge Acquisition*. Springer-Verlag.

Ford, K. M., Petry, F., Adams-Webber, J. R., & Chang, P. J. (1990, in press). An approach to knowledge acquisition based on the structure of personal construct systems. *IEEE Transactions on Knowledge and Data Engineering*.

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. (1986). Induction of inference rules for expert systems. *Fuzzy Sets and Systems*, 18, 315-328. Amsterdam: Elsevier Science Publishers.

Hunter, S., Stahl, H. A., & Novak, J. D. (1990). CMap: A Concept Mapping Elicitation Tool. Technical Report #90-006, Department of Education, Cornell University.

Jones, J. C. & Ford, K. M. (1990, in press). A neural net approach to knowledge elicitation from repertory grids. In M. Fishman (Ed.), *Advances in Artificial Intelligence Research*. (Vol. 2). Greenwich, CT: JAI Press.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*, Volumes 1 and 2. New York: W. W. Norton.

Kelly, G. A. (1965). The role of classification in personality theory. In B. Maher (Ed.), *Clinical Psychology and Personality*. New York: Wiley, pp. 289-300.

Kelly, G.A. (1970). A brief introduction to personal construct theory. In D. Bannister (Ed.), *Perspectives in Personal Construct Theory*. London: Academic Press, pp. 1-29.

Mancuso, J. C. & Eimer, B. N. (1982). Fitting things into sorts: The range corollary. In J. C. Mancuso & J. R. Adams-Webber (Eds.), *The Construing Person*. New York: Praeger, pp. 130-151.

Mischel, T. (1964). Personal constructs, rules, and the logic of clinical activity. *Psychological Review*, 71, 180-192.

Mitterer, J. & Adams-Webber, J. R. (1988). OMNIGRID: A program for the construction, administration, and analysis of repertory grids. In J. C. Mancuso & M. L. G. Shaw (Eds.), *Cognition and Personal Structure*. New York: Praeger, pp.91-106.

Novak, J. D. (1977). *A Theory of Education*. Ithaca, NY: Cornell University Press.

Novak, J. D. (1989). Helping students learn how to learn: A view from a teacher-researcher. Paper presented to the *Third Congress on Research and Teaching of Science and Mathematics*, Santiago de Compostala, Spain.

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

Tversky, A. (1977). Features of similarity. *Psychological Review*, 84, 372-352.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1990.

AN ARCHITECTURE FOR INTEGRATED KNOWLEDGE ACQUISITION SYSTEMS

Brian R Gaines

Knowledge Science Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4
gaines@cpsc.ucalgary.ca

ABSTRACT

An architecture for knowledge acquisition systems is proposed based upon the integration of existing methodologies, techniques and tools developed within the knowledge acquisition, machine learning, expert systems, hypermedia and knowledge representation research communities. Existing tools are analyzed within a common framework to show that their integration can be achieved in a natural and principled fashion. A detailed architecture for integrated knowledge acquisition systems is proposed that also derives from parallel cognitive and theoretical studies. An implementation of this architecture around a knowledge representation server is described.

INTRODUCTION

The past five years has seen an explosion in research on, and application of, knowledge acquisition methodologies, techniques and tools (Boose, 1989). The knowledge acquisition community world-wide has grown in numbers and scope of projects. There are significant international collaborative developments involving the sharing of ideas and software. However, the comments with which I began my paper at this meeting three years ago still apply:

"The problem now is not so much to access research and experience in knowledge acquisition but to make sense of the diverse and wide ranging material available and, in particular, to apply the results to improve effectiveness in the development of knowledge-based systems. There are a number of major impediments to such understanding and application:

- A diversity of techniques and tools that overlap in their applications and where it is not clear whether they are competitive alternatives or complementary partners.
- Lack of variety, detail and evaluation in the case histories of applications of the techniques and tools.
- Lack of access to the techniques and tools outside the narrow research communities originating them.
- Lack of standardization in the knowledge representation resulting from the techniques and tools making it difficult to integrate them and to interface them to existing knowledge based systems.
- Lack of standardization in the forms of data required by the techniques and tools making it difficult to apply them in the same situation and compare them.
- Lack of standardization in the user interfaces to the interactive tools making it difficult to integrate them in an effective environment for human-computer interaction.
- Lack of portability in the run-time environments required by the tools making it difficult to integrate them with other systems." (Gaines, 1987)

In 1987 I went on to say, "These are not unreasonable problems at this stage of development of knowledge based systems." In 1990 I would like to propose to this meeting that these problems are becoming highly unreasonable. We should accept them as the major research challenges for the 1990s, and develop an action plan to overcome them. They are challenges not only for us as individual researchers, research groups and organizations, but also for the knowledge acquisition community as a whole. Indeed many of the problems are essentially ones for the community not the individual—we can all contribute but no one of us alone can solve them.

This paper draws on the research and results of many researchers in the knowledge acquisition and knowledge representation communities to propose a practical architecture for future knowledge-based systems. It synthesizes a system design from what already exists, putting a diversity of well-founded and widely used approaches to knowledge acquisition within an integrative framework. The design is intended to be clean and simple, easy to understand, and easy to implement. It is very open in its architecture, providing a frame within which to experiment with new approaches to knowledge acquisition rather than a cage that locks us in to static paradigms. It is also intended to be practically applicable by being based on what we already know how to do well.

The open, international collaboration in the knowledge acquisition community is by far the most important phenomenon in current research. Ideas and technologies need to circulate freely in order to be in the right place at the right time—the place where the appropriate researcher can give them new impetus, and the time when he or she has the creative opportunity to do so. This paper is not about anybody's system or research product. It is a distillation of ideas from the whole community put forward as a contribution to a next generation of system designs. It encapsulates the objectives of many projects world-wide. I hope it will encourage the continuation and expansion of collaborative research.

KNOWLEDGE ACQUISITION PARADIGMS

This paper will adopt a techniques and tools perspective on knowledge acquisition. A companion paper (Gaines & Shaw, 1990) provides a more foundational cognitive science and logical perspectives, and appropriate links will be made to it. There are other significant perspectives that focus on methodological issues (Shaw & Woodward, 1989; Wetter & Woodward, 1990). If the tools are well designed, the foundational perspectives should provide theoretical justification for them, and the methodological perspectives should provide meta-knowledge guiding their use. Hopefully, the systematic design and presentation of the tools architecture will expedite the analysis of these deeper issues.

Four major paradigms underlie the most widely applied knowledge acquisition tools:

1 Direct editing of knowledge in a semantic network, frame, rule, representation

Graphic editors providing direct access to semantic network representations allowing knowledge to be encoded in frames and rules provide the most common development environment for knowledge-based systems. They are part of the application programming support environment of most expert system shells, and the widespread availability of modern graphic workstations has made it possible to provide excellent knowledge visualization environments.

A wide range of knowledge acquisition tools have been developed that structure and improve the graphic editing environment, often taking advantage of domain knowledge to provide a more specific, meaningful and familiar knowledge framework to the expert.

Examples are MOLE (Eshelman, Ehret., McDermott & Tan, 1987), KNACK (Klinker, Bentolila, Genetet, Grimes, & McDermott, 1987), SALT (Marcus, 1987), ONTOS (Monarch & Nirenburg, 1987), Cognosys (Woodward, 1990), KEATS (Motta, Eisenstadt, Pitman & West, 1988), CODE (Skuce, Shenkang & Beauville, 1989), CAMEO (Jones, 1989), and techniques directly supporting domain ontologies such as KADS (Breuker & Wielinga, 1987).

Figure 1 characterizes the major features of these direct editing systems. The expert interacts through a graphic interface with an underlying semantic network knowledge representation schema which may already contain pre-encoded domain knowledge. What is elicited are:

- The distinctions that the expert makes about domain entities (attributes and relations).
- The way in which these distinctions are grouped and constrained to form major concepts.
- The entailments between concepts that constitute procedural, decision-making rules in the domain.

Older systems have less well-structured knowledge representations but Figure 1 captures the essence of recent developments in knowledge representation that are moving towards very clean, and theoretically well-founded schema.

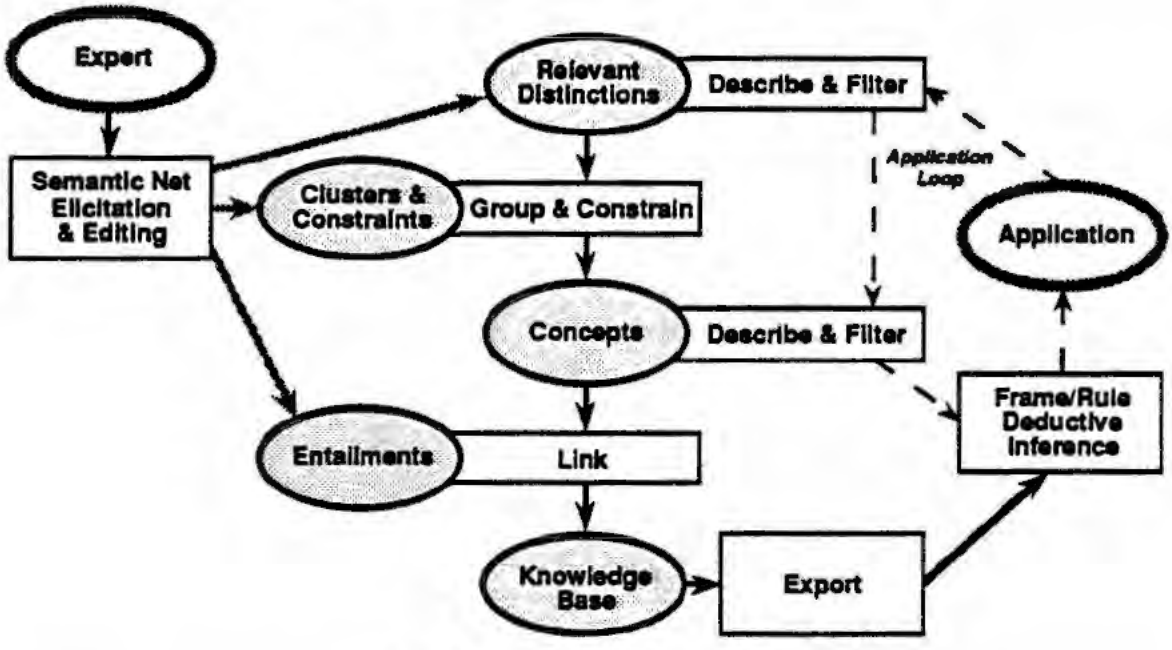


Fig.1 Semantic net architecture

The knowledge base of frames and rules developed in this way is then usually exported to a performance tool and validated against test case data. This generates the application loop shown on the right in which the expert's distinctions lead to a description of the problem which is structured through the concepts leading to the application of the inference rules that link them. Many acquisition tools also incorporate some form of performance tool so that this validation can be made part of the elicitation process.

The advantage of knowledge editing tools is that, if the knowledge is overtly available, they provide a fast and effective rapid prototyping environment for its direct capture. Other techniques assume that the nature of expertise is such that much knowledge is not so directly available. However, effective editing tools may be seen as essential bedrock to any integrated knowledge acquisition architecture. There will usually be significant

sub-domains where direct elicitation is possible, and provides the fastest, most effective means of knowledge capture. There is always a need to present and make available for validation and editing the knowledge structures captured by indirect methods.

Many indirect knowledge acquisition tools leave the knowledge presentation and editing to the associated performance tools since these often have excellent facilities. However, in an integrated architecture it is important to incorporate editors in the knowledge acquisition tool that interact effectively with all the different forms of knowledge captured. One of the major problems to be overcome is that once the knowledge has been exported and edited in the performance system it has lost its relation to the acquisition system. Most current knowledge acquisition tools do not support long-term development and knowledge base maintenance largely because of this lack of integration.

2 Inductive derivation of knowledge from data sets of varying quality

When experts cannot directly enter a knowledge structure emulating their expertise, they may still be able to point the knowledge engineer towards case histories that incorporate that expertise in being described in terms of relevant attributes and correct decisions. Empirical induction techniques may then be used to derive knowledge structures underlying the decisions made in these cases (Michalski & Chilausky, 1980).

The best known empirical induction methodology is that of Quinlan's ID3 which has been refined in many ways, particularly to tolerate noise (incorrect decisions), resulting in the latest implementation, C4.5 (Quinlan, 1987). The original decision tree structure of ID3 based on a subsumption hierarchy of concepts with rules at the leaf nodes only is unnecessarily large in many situations, and extensions to ID3 have been developed that generate modular production rules directly, such as Cendrowska's (1987) Prism. This also can be extended to deal with noisy data as in Induct (Gaines, 1989).

The extensions that deal with noisy data also make it possible to combine the decision tree and modular rule methodologies to generate default reasoning in which rules are placed at non-leaf nodes in the subsumption structure, and more specialized rules override more generalized ones. Such default rule structures are more compact than either decision trees or modular rules—they are generated by both C4.5 and Induct.

Figure 2 shows the minimal default rule structure generated by Induct for Cendrowska's (1987) contact lens problem—the notation is that from Gaines & Shaw (1990)—ellipses denote concepts, black arrows denote the subsumption relation, and gray arrows rules. What is significant about this diagram is that it illustrates the way in which empirical induction techniques can be used to generate conceptual structures for a domain. The left hand and right hand sides of a rule each defines a concept. With default rules in particular these give rise to a natural subsumption ordering. Thus the distinctions, concepts and entailments shown directly entered in Figure 1 can also be indirectly generated through induction as shown in Figure 3.

Figure 3 characterizes the major features of conceptual induction systems. The expert indicates a database whose designer has supplied distinctions to categorize the world and cases described in terms of these distinctions to represent it. What is derived are:

- That subset of the distinctions that are relevant to the decisions.
- The way in which these distinctions are grouped and constrained to form concepts.
- The entailments between concepts that regenerate the decisions in the database.

Classic empirical induction tools do not generate the conceptual structure but this is a fairly simple extension.

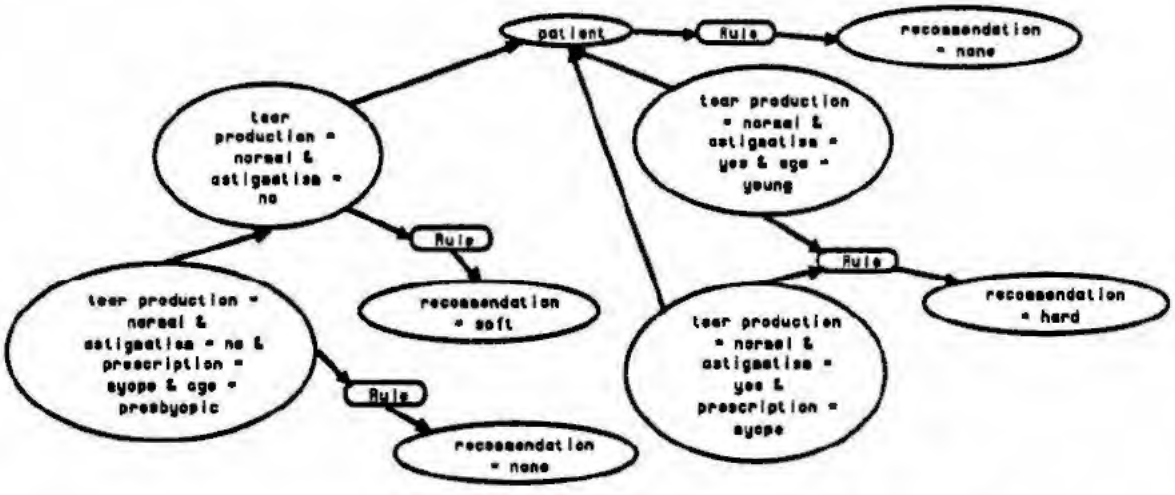


Fig.2 Conceptual structure generated through empirical induction

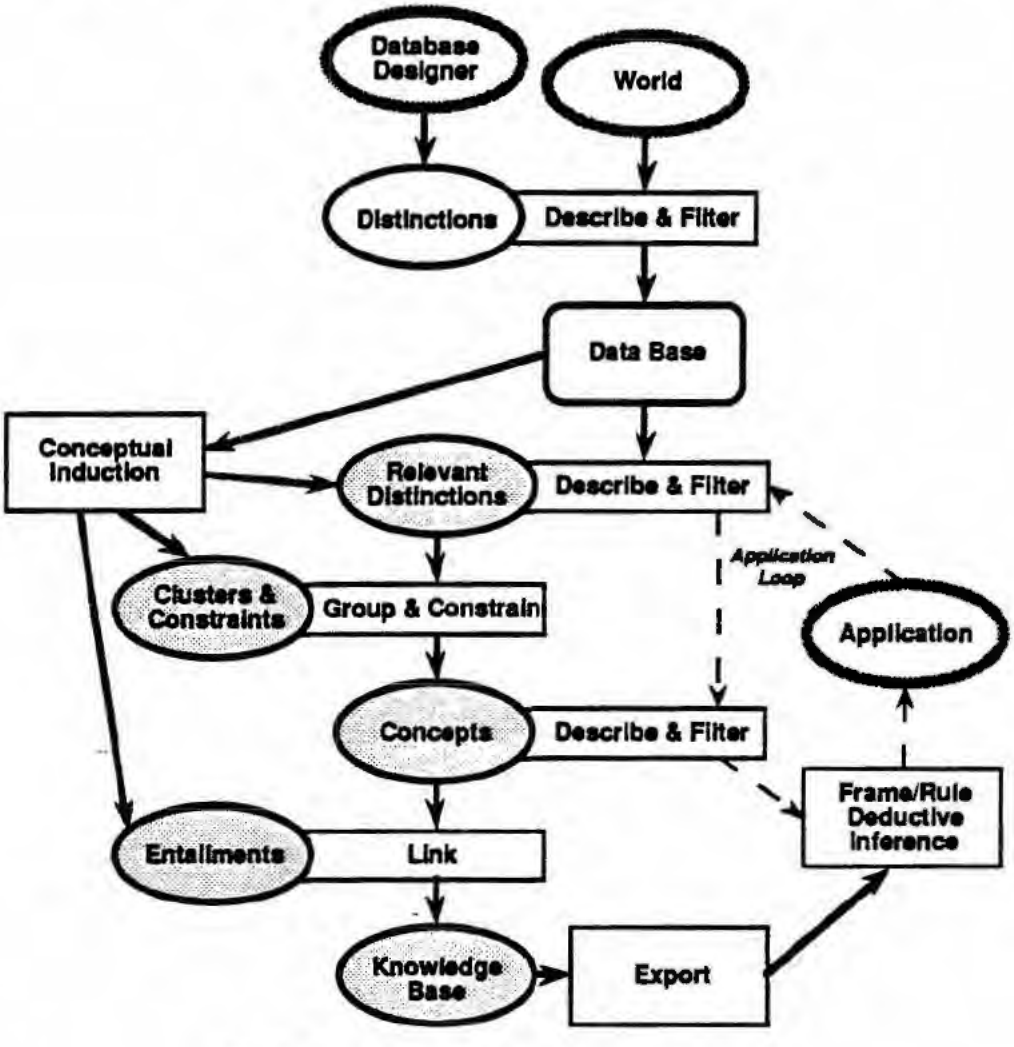


Fig.3 Conceptual induction architecture

Figures 1 and 3 indicate strong similarities between the outcomes of direct knowledge elicitation and conceptual induction. This is as it should be since both techniques are building a complete knowledge base. However, what is not apparent is the relative efficiencies of the two methodologies—that is, how large a database is required to generate the knowledge required to solve a problem? Figure 4 shows the results of one study to investigate the relationship between empirical induction and expertise transfer as knowledge acquisition methodologies (Gaines, 1989). Cendrowska's contact lens data was subjected to random distortion with known statistics to generate large datasets with a certain number of irrelevant binary attributes and a certain percentage of incorrect decisions. Induct was then run on the dataset with 5,000 items, 4,999 items, and so on, until the dataset failed to generate rules giving correct performance. This was done ten times for different datasets of the same type to give estimates of the mean and standard deviation of the size of dataset required to generate correct performance for different forms and levels of distortion.

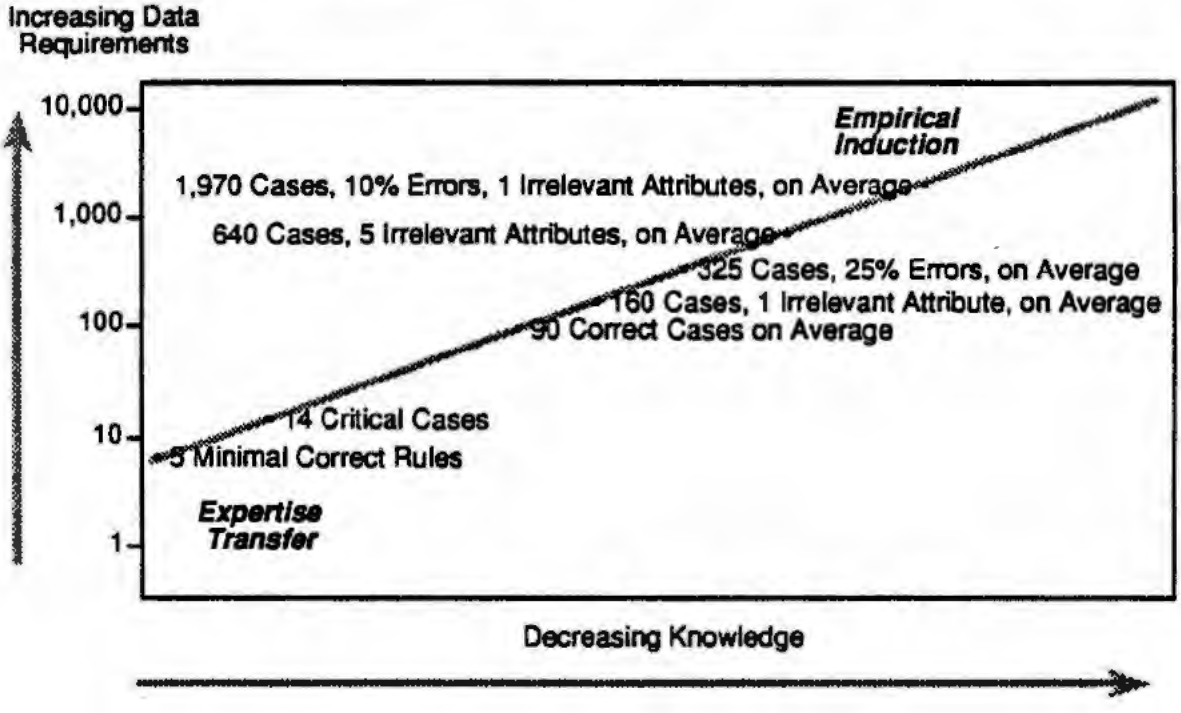


Fig.4 Data/knowledge tradeoff relating expertise transfer and empirical induction

The results shown in Figure 4 indicate the very wide range of the tradeoff between data and knowledge: from direct entry of the minimal knowledge structure of 5 default rules; through entry of 14 critical cases; through an average of 90 randomly selected correct cases; to 325 cases with 25% errors; 640 with 5 irrelevant binary noisy attributes; to 1,970 when a single irrelevant attribute interacts with a 10% error rate.

The moral from Figure 4 is not that expertise transfer is better than empirical induction, although the direct entry of overt knowledge is clearly highly ergonomic if it is available. It is rather that both techniques are capable of producing equivalent quality knowledge, and there is a continuum between them in which knowledge is traded for data. The knowledge of the database designer is implicit in the distinctions made and data selected, and can be made explicit through induction. In practice, it is rare to be working at either extreme of this curve in system development—the repertory grid techniques described in the next section focus on the middle region, attempting to elicit relevant distinctions and critical cases, and building conceptual structures and rules by analyzing these.

3 Indirect elicitation through critical cases described in relevant attributes

Repertory grids are another technique for knowledge elicitation when experts cannot directly enter a knowledge structure. They stand between direct editing and empirical induction, and prompt the expert for distinctions relevant to the problem domain and for critical cases that exhibit significant phenomena in the domain. The prompting is done through online analysis of the data being entered leading to feedback to the expert suggesting missing distinctions and cases. This highly focused feedback aids the expert in developing his or her mental model of the domain. It also reduces the inefficiencies of duplication and the mental blocks of psychological set, supporting rapid prototyping.

A wide range of knowledge acquisition tools have been developed that incorporate repertory grid elicitation and analysis as their major interface to the expert. Examples are PLANET (Shaw & Gaines, 1983), ETS (Boose, 1984), AQUINAS (Boose & Bradshaw, 1987), KRITON (Diederich, Ruhmann & May, 1987), KITTEN (Shaw & Gaines, 1987), and KSSO (Gaines & Shaw, 1987).

Figure 5 characterizes the major features of repertory grid elicitation systems. The expert interacts through a graphic interface to enter individuals in the domain (elements) and bipolar distinctions (constructs). Conceptual clustering techniques are used to feedback the elicited domain structure in an easily assimilated form for validation. Rule induction is used to generate entailments, or, more recently, conceptual induction as in Figure 3 to generate a default rule structure. What is elicited are:

- The distinctions that the expert makes about domain entities.
- Critical cases exhibiting the major phenomena affecting decision-making in the domain.
- The way in which distinctions are grouped and constrained to form concepts.
- The entailments between concepts as induced from the critical cases.

Older systems did not have explicit conceptual induction but left grouping into concepts or frames as a task for the export module.

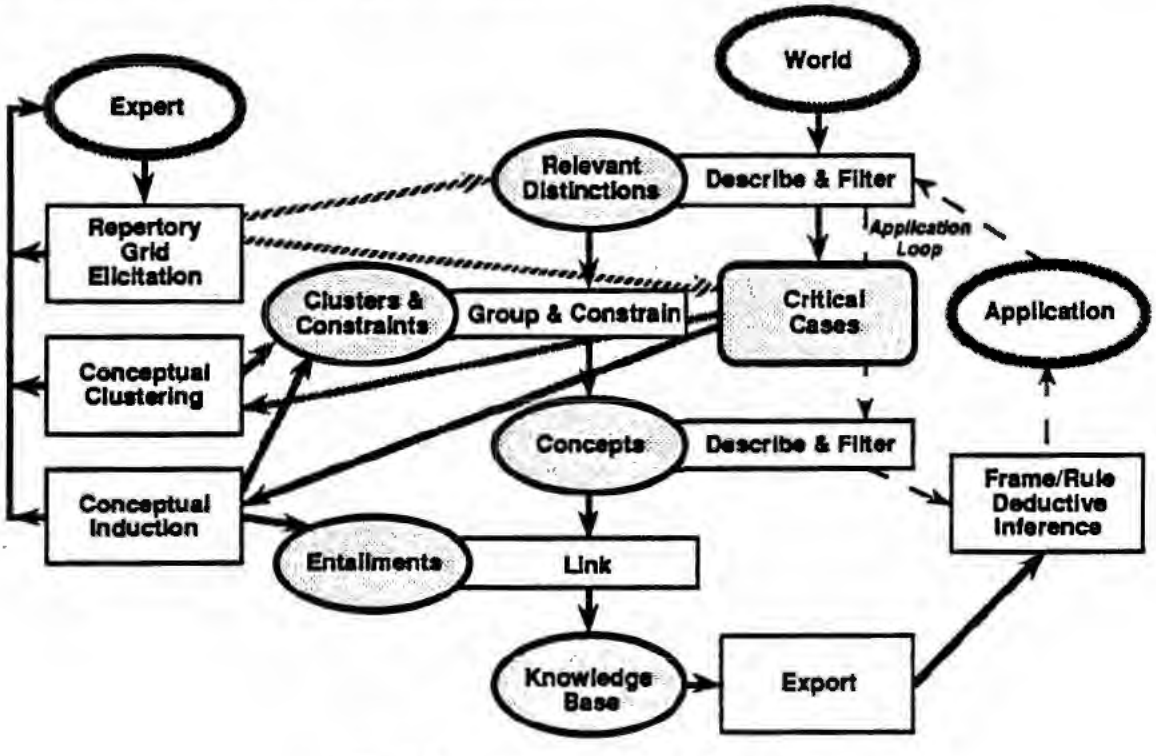


Fig.5 Repertory grid architecture

The clustering and induction modules in Figure 5 are extensions of the basic repertory grid technique incorporated in PLANET and KSS0, and other major extensions have been incorporated in other tools. In particular, AQUINAS makes provision for a wider range of data types than the rating scales of the basic grid, and also allows hierarchies of cases and attributes to be specified that are related to those of semantic nets. Both AQUINAS and KSS0 also incorporate tools supporting multiple sources of expertise and analyzing the relationships between different sources.

There are also other indirect methods of knowledge elicitation that have been used to enhance interaction with direct editing tools. For example, Cognosys analyzes protocols in terms of Graesser and Clark's (1985) linguistically derived "general knowledge structures", and implementations of the KADS methodologies also provide general ontological frameworks directing the expert and knowledge engineer's model building.

4 Semi-formal elicitation and structuring through hypertext and hypermedia

The three techniques so far described focus on the elicitation, derivation and structuring of computational knowledge, that is, knowledge that can be operationalized within the computer as an active emulation of expertise. However, much knowledge is informal yet still valuable in an advisory expert system—text and pictures can encode and impart usable expertise, supplementing computational knowledge. Thus, the parallel development of hypertext and hypermedia is having a substantial impact on expert system architectures and knowledge acquisition tools.

Figure 6 shows some of the features of modern document processing systems that impinge on knowledge acquisition. Documents may be acquired from many sources, displayed, parts re-used in other documents, and parts linked for hypertext navigation. The text in documents may also be analyzed for associative cluster and a person may group these clusters to indicate significant concepts—such text analysis has been incorporated as a preliminary acquisition stage in repertory grid tools, for example the TEXAN module in KITTEN and KSS0.

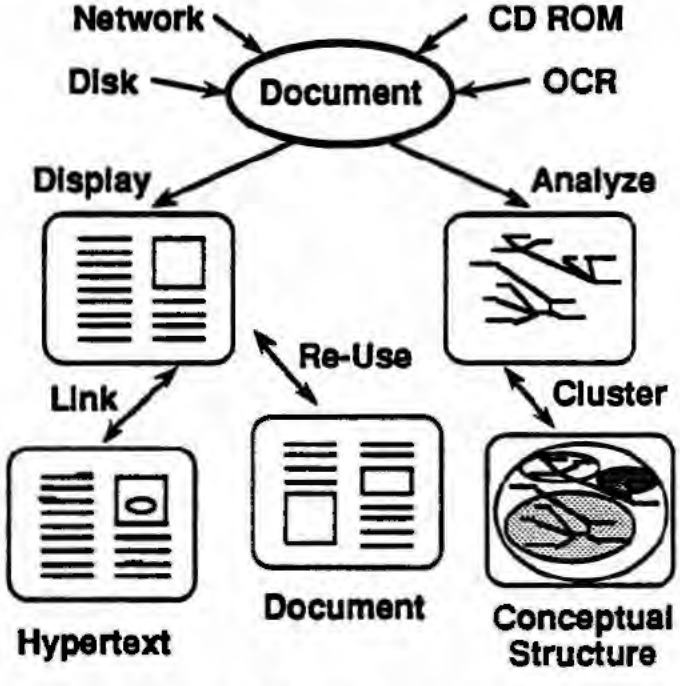


Fig.6 Hypermedia and text processing architecture

Hypertext systems have been coupled to knowledge acquisition tools to provide annotation of the distinctions made and cases described which can then be used to provide explanation facilities in the final performance system (Gaines, Rappaport & Shaw, 1989; Gaines & Linster, 1990). Apple's HyperCard system also provides a powerful programming language and user interface tools which has made it an attractive environment in which to build knowledge acquisition tools such as Cognosys and CAMEO. These then naturally incorporate many hypertext techniques. Other knowledge acquisition tools such as KEATS have been built around a hypertext environment specifically designed for knowledge acquisition.

There is also knowledge acquisition and linguistics research targeted on the direct transfer of knowledge expressed in text to structures of frames and rules (Hausser, 1989; Gomez, 1990). Since so much knowledge is already overtly encoded in text and diagrams, in the long term this will become an essential knowledge acquisition technology. However, it is still at an early stage of development that does not satisfy the "what we already know how to do well" criterion, and is not yet suitable for incorporation in integrated systems.

AN INTEGRATED SYSTEM ARCHITECTURE

The previous section has analyzed the current major knowledge acquisition techniques and tools within a common framework to show their similarities and differences with a view to designing an integrated system in which they operative effectively together. A companion paper analyzes the cognitive psychology and algebraic structure of knowledge representation to develop foundations for knowledge acquisition systems (Gaines & Shaw, 1990). Practice and theory join together in the knowledge acquisition architecture shown in Figure 7 which is an extension of Figure 14 in the other paper to incorporate the acquisition of informal knowledge from interviews, protocols and media as a stage ⑨.

Stage ① consists of using the informal knowledge to elicit the major coherent sub-domains that together encompass the significant phenomena in the overall domain. Stage ② consists of using the repertory grid methodology in each sub-domain to elicit relevant attributes and critical cases. Stage ③ consists of using the conceptual induction methodology to derive concepts and rules from the grids. Stage ④ consists of linking the sub-domains together, generally by specifying as a constraint in one domain that some role in that domain has as value an individual in another. Stage ⑤ consists of testing the overall knowledge base and iterating back through any of the earlier stages in order to develop and refine it further.

The architecture of Figure 7 is intended to be illustrative of the way the various knowledge acquisition techniques described naturally combine than a rigid framework for an integrated tool. For example, some sub-domain structures may be developed by direct editing, others by induction from databases, others by text analysis, and others by tools not yet defined or developed. The point is that that the techniques and tools that we have already come together naturally in this framework, and it also provides an open architecture for incorporating further techniques and tools. The companion paper also gives strong theoretical foundations for what is shown. In the longer term, the architecture shown may need major extension, but currently it seems to encompass what is available to us in knowledge acquisition.

Figure 8 combines and extends Figures 1, 3, 5 and 6 to show the modules necessary to support the architecture of Figure 7. Nearly all the modules shown have been described already. At the upper left, text and picture entry and analysis are shown as prompting the expert and developing a hypermedia base that also has links to the computational knowledge structures.

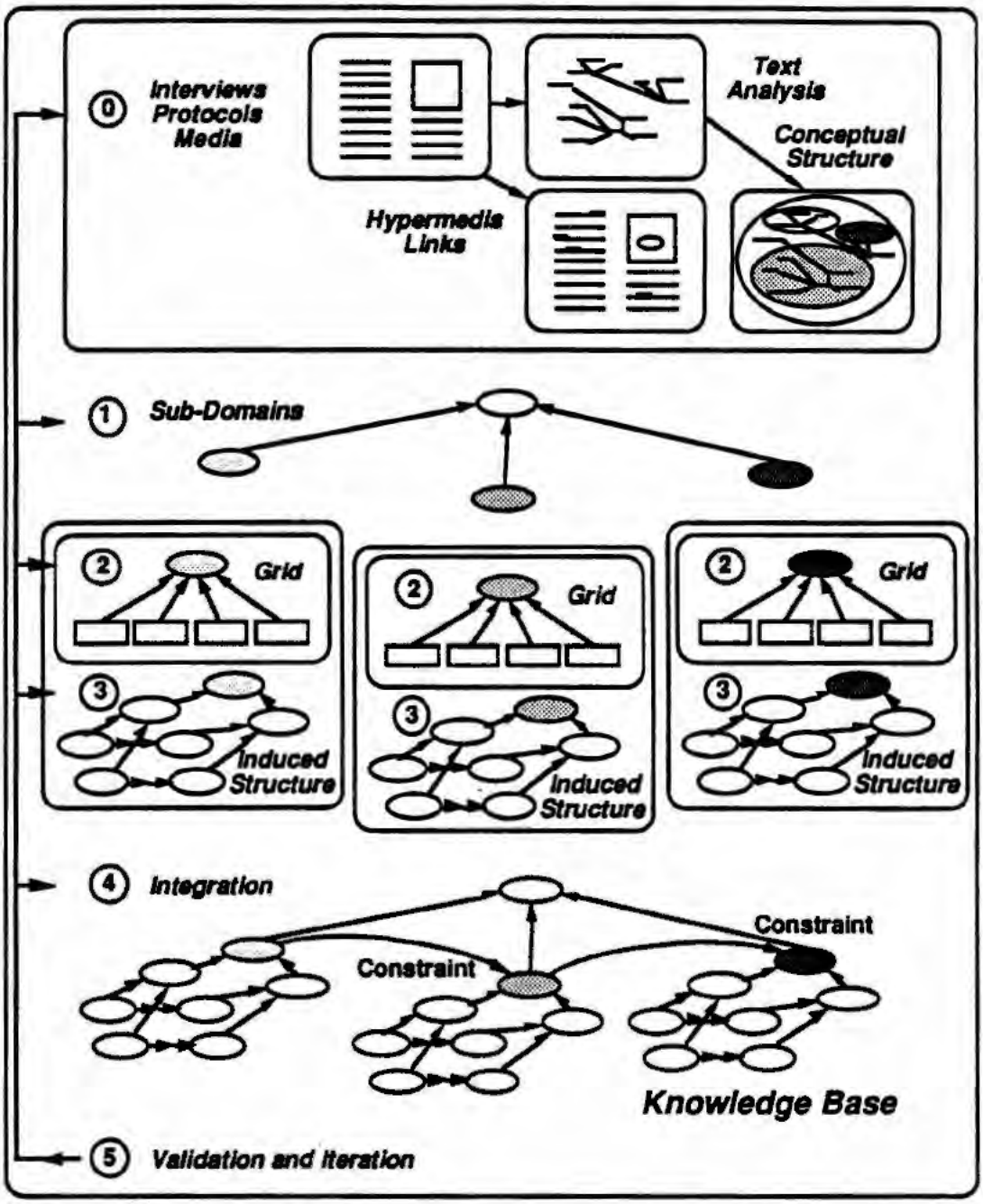


Fig.7 Integration of knowledge acquisition techniques

At the lower right of Figure 8 the knowledge base has been separated into a procedural one based on the rules and a declarative one based on the cases, and two additional inference paradigms have been added. The center one combines frame/rule inference with *case-based* inference, as, for example in PROTOS (Bareiss, 1989) The lower one provides *analogical* inference (Leishman, 1989) based on the cases in the declarative knowledge base, as, for example, in AQUINAS' performance module. These mixed paradigms based on both abstract knowledge and concrete cases are seen as the target inference modes. Note that the inference modules are now shown as an integrated part of the knowledge acquisition system for validation purposes, but that the need to export to specialist performance systems is still recognized.

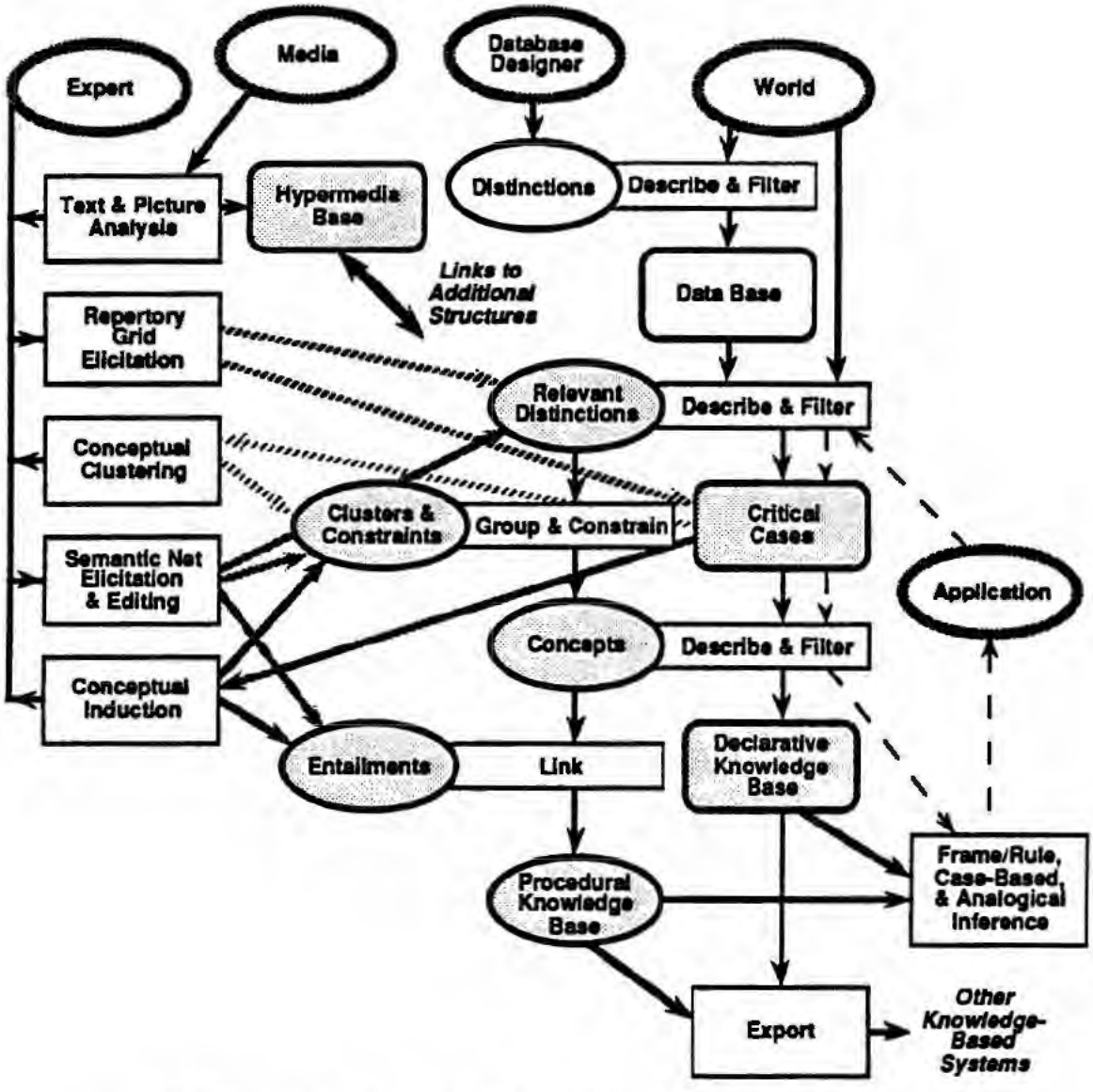


Fig.8 Integrated knowledge acquisition system architecture

Even though inference is integrated, at the bottom right of Figure 8 an export module is shown since transfer of knowledge bases to specialist performance systems is still seen as an important requirement. We need to integrate performance tools with knowledge acquisition tools for testing and validation, and for long-term support and updating. However, there will continue to be major requirements for stand-alone knowledge acquisition tools and stand-alone performance tools optimized for their primary tasks.

This presentation has emphasized that the methodologies, techniques and tools necessary to implement the integrated knowledge acquisition system architecture shown in Figure 8 already exist in the knowledge acquisition, machine learning, expert system, and hypertext communities. There is one basic technology underlying Figures 7 and 8 that is significant to those communities but has been sufficiently important and self-contained to be the focus of a major community in its own right, and that is knowledge representation. One can view Figure 8 as a set of related applications modules around a very comprehensive *knowledge representation module*. This is proposed as an implementation paradigm in the following sections.

IMPLEMENTATION—A KNOWLEDGE REPRESENTATION SERVER

In recent years there have been major advances in the the theory and practice of knowledge representation systems originating from semantic nets. In particular the series of knowledge representation schemas and tools commencing with KL-ONE (Brachman & Schmolze, 1985), developing through KRYPTON (Brachman, Gilbert & Levesque, 1985) and currently culminating in systems such as CLASSIC (Brachman, Borgida, McGuinness & Resnick, 1989; Borgida, Brachman, McGuinness & Resnick, 1989) and LOOM (MacGregor, 1988) has reached a maturity of technology which offers the promise of knowledge representation 'utilities' or 'services' in Levesque's (1984) terminology. The logical foundations of the subsumption relation, techniques for its correct and complete calculation, and the interaction between representation power and the tractability of subsumption algorithms have been widely studied (Brachman & Levesque, 1984; Nebel, 1988; Schmidt-Schauss, 1989) and are becoming reasonably well-defined.

It is now feasible to develop knowledge representation servers on a par with floating-point arithmetic units and numeric libraries, as software (and perhaps ultimately hardware) modules with well-defined functionality and fast, reliable performance. As with arithmetic units, such servers by no means solve all the problems of a particular application domain, but they do greatly reduce the burden of system development, allowing effort to be focused on the specifics of particular systems rather than on what should be general utilities.

In view of these advances in knowledge representation, and the central role played by knowledge representation in knowledge acquisition, it seems reasonable to implement the system shown in Figure 8 around a knowledge representation server satisfying the following requirements:

- A representation schema that is theoretically well-founded, adequate to support current knowledge-based system applications, and simply extensible to cope with future requirements.
- A knowledge base definition language mapping unambiguously into the representation schema, supporting data interchange between knowledge acquisition systems, expert system shells, databases, and other forms of knowledge support systems.
- Implementations that are light-weight in their space and time requirements, portable across platforms, cleanly structured so that functionality, data structures, and algorithms are visible and easily checked, and open architecture with well-defined interfaces to external modules.
- Offering a range of basic service modules such as parsers, graphers and network communications.

Figure 9 shows the structure of an integrated knowledge acquisition system implemented as a set of modules communicating with a knowledge representation server. It is designed to manage multiple knowledge bases and to interface to other servers either of the same or differing architectures. The various sub-modules attached to the server are options offering specific additional functionality. The modules are (reading clockwise from the top left):

- Interface modules to other formal knowledge bases and KR servers, including databases.
- A hypermedia module allowing informal knowledge structures in text and images to be captured, accessed and linked. The linkage structure itself is held as a knowledge base.
- A text analysis module allowing documents to be analyzed in terms of word usage, and associations between significant words to be graphed—based on TEXAN in KSSO

(Gaines, 1988; Gaines & Shaw, 1989). This enables protocols and technical documents to be used to initiate knowledge acquisition.

- A repertory grid expertise transfer module allowing graphic definition of concepts and graphic creation and editing of individuals—based on the elicitation screens of KSS0.
- A conceptual clustering module allowing interactive definition of new concepts—based on the hierarchical and spatial clustering from KSS0.
- A knowledge editing module allowing the interactive development and editing of knowledge structures through a visual language.
- A conceptual induction module creating rules about specified subsets of individuals and transforming them to a minimal set of concepts and default rules—based on the Induct algorithm (Gaines, 1989).
- A problem solving module supporting frame, rule and case-based inference from the knowledge structures.
- A grapher laying out specified parts of the concept subsumption graph, concept structures and individual structures—based on an incremental layout algorithm by Watanabe (1989) that can be used interactively to support the production of clear visual knowledge structures.
- A language interface accepting and generating definitions and assertions in formal knowledge representation languages, both textual and visual.

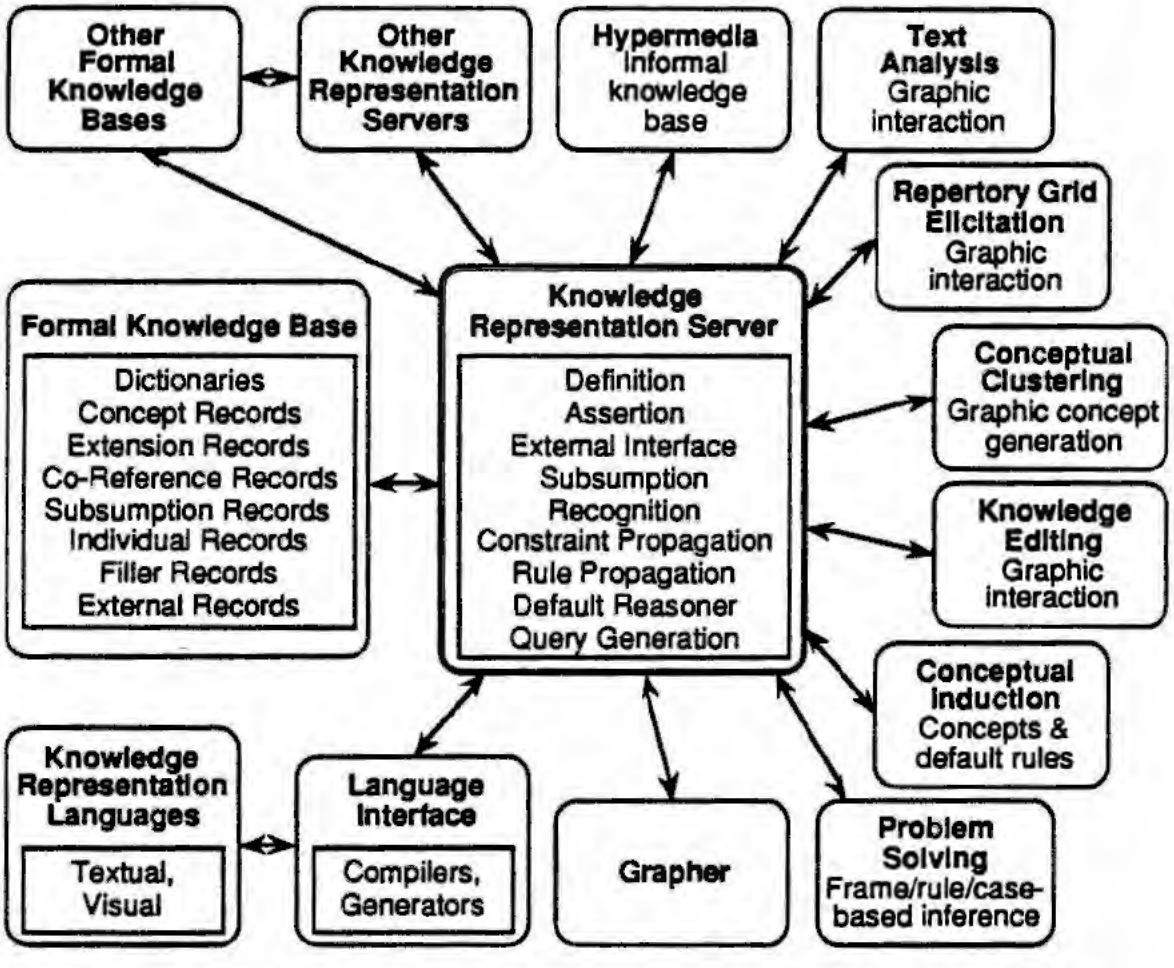


Fig.9 Knowledge representation server and associated modules

The data structures of the knowledge base and the functionality of the server are discussed in more detail below. The functionality of most of the modules on the right has already been described. The graphic interfaces and algorithms for eliciting, clustering, editing and induction modules already exist in tools such as KSS0 and AQUINAS—they have to be extended to a wider range of data types. The hypermedia informal knowledge base uses the server to maintain its linkage structures. The links to other servers and knowledge bases reflect the open architecture requirement. The problem solving module using frame, rule and case-based inference is intimately connected with knowledge processing functions within the server and is discussed below. The grapher is a general layout system to show the concept subsumption graph, concept and individual structures, and to provide interactive access to these structures. The language interface to knowledge representation languages reflect the knowledge base definition languages requirement.

There are a number of contenders for a knowledge representation server already within the knowledge acquisition community such as ONTOS (Monarch & Nirenburg, 1987) written in Lisp and CODE (Skuce, Shenkang & Beauville, 1989) written in Smalltalk. We have developed an implementation of a CLASSIC-like (Borgida, Brachman, McGuinness & Resnick, 1989) system in a dialect of C++—choosing the schema because it is the latest incarnation of KL-ONE, designed to be computationally tractable, and extremely well-defined—and choosing object-oriented C for speed, portability and the existence of very good class libraries. The purpose of the implementation is to investigate feasibility rather than to create a definitive server. The initial CLASSIC implementation took some six person-weeks and the extension of the schema to include correct default reasoning correctly tracking assertions and retractions about the same time again. Modern programming technology makes it straightforward to create open architecture knowledge representation servers in a range of languages running on a variety of platforms.

Figure 10 shows the objects used in the server implementation and their data structures. Our basic class library provides variable-length record objects which are sub-classed to provide dictionaries, sets and record objects for concept and individual records. The objects are:

- *Concept records* hold the constraint defining a concept: its extensions as the subsets of individuals that may be, and must be, included in any subset recognized; its cardinality and co-reference constraints on any subset recognized; its rule concept, if any, which will be asserted of any individual recognized; and, for each role constrained in the concept, the concept constraining it.
- *Individual records* hold the templet concept defining intensional constraints upon an individual and, for each role constrained in the templet or filled in the individual, a specification of the role fillers.
- *Filler records* hold subsets of individuals that fill slots.
- *Dictionaries* recognize concept, role and individual names and generate an accession number used to index other structures. Syntactically there is a clear separation between concepts, roles and individuals and a lexical item can occur in more than one dictionary with different meanings.
- *Extension records* hold subsets of individuals, typically those that may occur in a role filling set and those that must occur in it.
- *Data records* hold subsets of external individuals that are explicitly recognized by the server, notably integers, reals, dates and strings.
- *Co-reference records* hold equivalence subsets of role chains expressing co-reference constraints.
- *Subsumption records* hold computed subsumption relations on a three-valued basis, that one concept subsumes another, does not, or cannot.

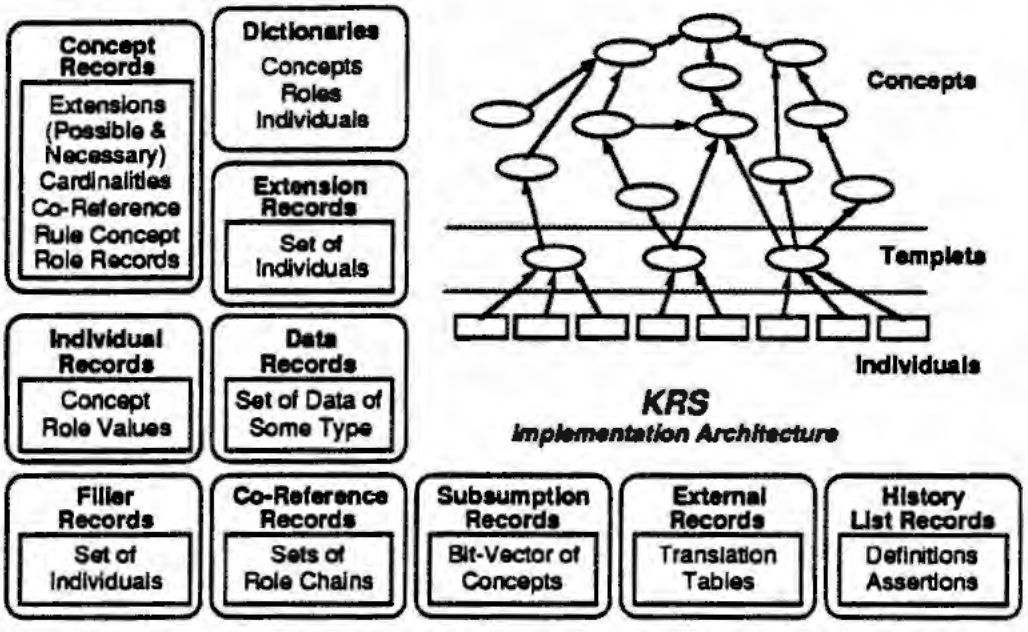


Fig.10 Knowledge representation server data structures

- *External records* hold tables translating between internal and external concepts and individuals.
- *History list records* keep track of explicit and implicit definitions and assertions in such a way that the knowledge base may be rolled back to any previous state.

A number of optimizations are applied in the implementation to minimize storage and maximize speed. For example, special codes are used for the empty, universal and singleton sets such that the set objects never store them, and subsumption is calculated in such a way that only a fast lookup is needed to check subsumption between role concepts rather than a recursive call. The object implementation allows these optimizations to be put in override methods in sub-classes so that the kernel implementation is small and easily checked but potentially slow, and the full implementation can be checked against the kernel for correctness and optimized for speed.

The server satisfies the requirements for knowledge representation for knowledge acquisition so far defined—the co-reference constraints go beyond them but are necessary to support analogical reasoning (Leishman, 1989), and, in particular, Russell's (1989) *determinations*. It provides a number of functions:

- A space- and time-efficient post-relational database for variable-length records of variable-length data whose types include: other records (relations); strings; integers; reals; dates and times; and sets of these.
- A constraint-specification system that allows intensional constraints on slot-fillers in records to be specified (that is, concept definitions).
- Automatic tracking of relations between systems of constraints—the *subsumption* relation of strict inheritance, and *disjoint* relation of attribute formation.
- Intensional constraints may be specified not only in general but also for individual records—this enables 'abstract objects' to be created which shadow concrete instances.
- Automatic propagation of intensional constraints on individual records as the individuals are specified as slot fillers in other records.
- Automatic tracking of whether an individual record satisfies the constraints defining a concept—this enables individual records to be 'recognized' as falling under a concept even if they have not been specified to do so.

- Sufficient conditions for an individual to be recognized by a concept may be associated with necessary conditions that then hold—this enables 'rules' to be specified.
- The failure of an individual to be recognized by a concept is classified as 'factually necessary' (due to slot fillers), or 'open' (further data could lead to recognition)—this allows principled default reasoning.
- The detailed basis for the failure of an individual to be recognized by a concept is available—this allows TEIRESIAS-like debugging of the knowledge base.
- A distinction between intensional constraints and assertions (facts)—the propagation of constraints from slots, or through rules, never changes individual records, only the abstract objects constraining them—this allows facts to be distinguished from inferences based on them and supports proper updating of inferences when assertions are retracted.

The algorithms providing these functions within the server are logically complete and correct—there are no heuristics. This is important in testing and validating the knowledge structures—inferences made with test cases can be traced precisely to the underlying conceptual structures. When exporting the knowledge base to shells that use heuristics such as 'first rule fired' or 'rule priorities' to simulate default reasoning the test cases must be re-validated.

A number of the modules shown in Figure 9 have already been implemented around the server. A recursive descent compiler provides a knowledge representation language similar to that of CLASSIC. A grapher has been implemented that supports interaction through popup menus, including calls to HyperCard. KSS0 has been interfaced to the server through an inter-application protocol so that its graphic elicitation, clustering and induction modules may be used to generate knowledge structures in the server. The problem-solving module currently fully supports inferences based on the procedural knowledge, the concepts and rules. We are investigating some interesting case-based decision-making models that use only operations such as specialization and generalization available within the server. It is interesting to discover how much can be achieved within the representation schema without adding other calculations and heuristics in external modules.

EDITING KNOWLEDGE STRUCTURES

KL-ONE-style knowledge structures are notoriously difficult to edit because they are based on the 'top-down' definition of general concepts and their stepwise refinement to more specific concepts. This is reasonable in well-established domains where the ontology has been developed as part of a professional discipline, but it is rarely a suitable basis for knowledge acquisition in novel domains. Experts often develop their knowledge structures 'bottom-up' from cases through commonality to general ontologies, and find a 'top-down' approach very unnatural. There are a number of ways of overcoming this problem—repertory grid, conceptual clustering and induction tools develop concept hierarchies 'bottom-up' from cases—but general concept editing services are difficult to define.

The problem is to design an interface allowing the person editing to adequately express his or her intentions. For example, if a concept is edited it is simple enough to maintain referential integrity by disallowing use of concepts defined after it in the definition, and even this constraint can be relaxed in a more intelligent editor that places a new or edited concept in an appropriate place in the referential order if one exists. However, it is difficult to determine whether the user intends, when editing, that subsumption relations be preserved and hence that edits should be propagated, and, if so, whether the user really wants full propagation or the option of checking what is occurring.

The BLIP knowledge acquisition tool developed at the Technical University of Berlin to support 'sloppy modeling' (Morik, 1987) provides a conceptual system development and editing tool that computes all consequences of definitions dynamically, including induction from cases. For the initial stages of knowledge acquisition even this may be too strong in that experts often generate inconsistent knowledge structures but are not ready to resolve the inconsistencies until they have a developed a fairly large-scale model giving them an overall perspective on the task and domain. The graphic knowledge editing tool shown on the center right of Figure 11 supports this relaxed approach. 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 ontology '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.

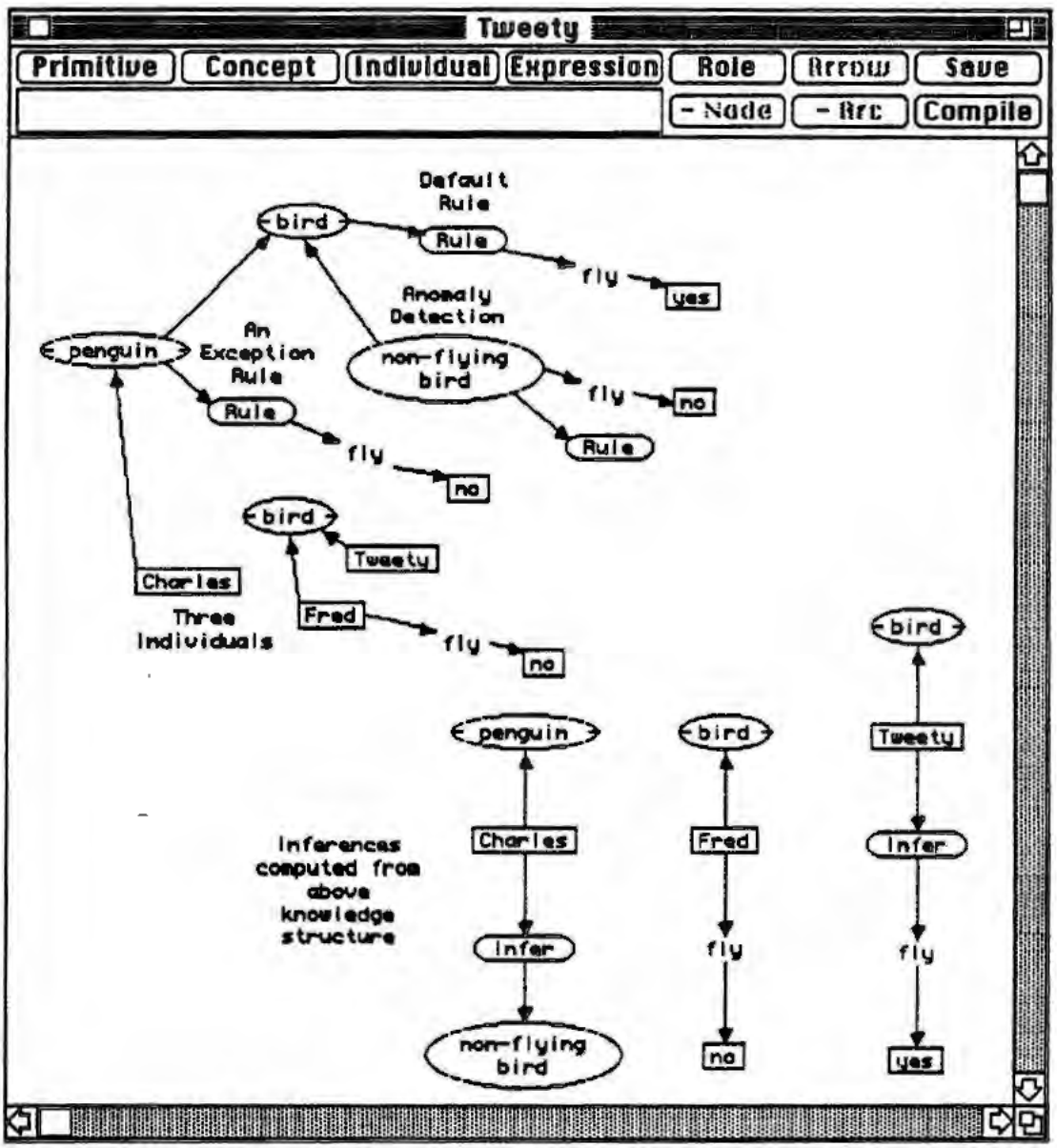


Fig.11 Graphic knowledge editor for the knowledge representation server

The knowledge structure at the top left of Figure 11 is an excessively classic one in artificial intelligence, but it still exemplifies significant features of knowledge acquisition and reasoning. A three rule default reasoning structure is shown in which birds by default can fly, but penguins are an exception, and birds that factually cannot fly are an anomaly. Three individual instances of birds are graphed at the middle left. When this structure is compiled for KRS it generates the knowledge base shown in Figure 12. When this is run for inferences and the individuals re-graphed the result is the output on the lower right.

The visual language used in Figure 11 is precisely defined. Concepts are ovals, individuals rectangles, roles (and annotation) unboxed text, and expressions rounded-corner boxes. The interpretation of the arrows is well-defined by the types of the objects at their head and tail. Arrows with open heads denote rules. Lines without arrows connecting concepts would denote that the concepts are disjoint. The 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. Role objects are only noted when on paths and hence may also be used for annotation. The text analysis module at the top of Figure 9 generates such 'annotation' which may then be developed into a formal graphic specification by specifying objects as concepts and individuals.

```
Primitive(bird)
Concept(non-flying bird, bird, (All fly,  (Include no)))
Primitive(penguin, bird)
Rule(bird, (All fly,  (Include    yes)))
Rule(non-flying bird)
Rule(penguin, (All fly,  (Include no)))
Individual(Tweety, bird)
Individual(Fred, bird, (Fills fly, no))
Individual(Charles, penguin)
```

Fig.12 Visual knowledge base icompiled for knowledge representation server

The accompanying paper (Gaines & Shaw, 1990) gives a more substantial example in terms of the minimal five rule default reasoning solution of Cendrowska's (1987) contact lens prescription problem.

DEFAULT REASONING

One important application of the server in knowledge acquisition is to support the restructuring of rules inductively derived from elicited datasets as a minimal set of concepts and rules that can be presented to the expert in an easily understood form for checking and validation. This involves default reasoning since the best understood presentation is usually the most commonly occurring situations as the major rules with the rarer situations as exceptions. CLASSIC in its current definition supports rules but not default reasoning, and there are known problems to simply incorporating exceptions as rules triggered by concepts lower in the subsumption hierarchy (Ginsberg, 1987).

KRS fully supports default reasoning in that rules are checked by tracing the subsumption hierarchy and giving precedence to inferences made at lower levels. Thus, in the knowledge structure defined in Figures 11 and 12, the top level default rule is that an individual recognized as a 'bird' has 'fly' set to 'yes.' This is superseded by 'no if the individual is recognized by the subsumed concept, 'penguin.'

This reasoning is logically correct if all the facts about the individual are known. It only becomes default reasoning if the exception rule cannot be tested because some facts are unknown. KRS can indicate whether an inference based on partial information is plausible or exact because it computes a three-valued subsumption relation which allows the system to recognize when no further assertions could make an exception rule trigger. This is termed a 'release' and may be viewed as a rule which triggers when it is impossible for an individual to satisfy a concept except through retraction. Releases may be used to manage the frame problem by discriminating between plausible reasoning based on defaults and exact reasoning based on adequate information.

The reasoning algorithm in KRS is efficiently implemented and operates rapidly even with large knowledge bases; for example, the Garvan Institute of Medical Research thyroid diagnosis knowledge base, GARVAN-ES1, consists of 650 rules (Compton & Jansen, 1989). The 'ripple-down rules' version of this was moved to KRS and checked on the cornerstone case database of some 650 cases, taking less than one second a case on an Apple Mac IIci. Nebel's (1990) 'worst-case' example of the combinatorial explosion of anonymous concepts runs without problems—for example, his 20 concept example generates a total of 1,701 concepts and a total data structure of 460 Kbytes.

CONCLUSIONS

An architecture for knowledge acquisition systems has been proposed that is based upon the integration of existing methodologies, techniques and tools developed within the knowledge acquisition, machine learning, expert systems, hypermedia and knowledge representation research communities. Existing tools have been analyzed within a common framework to show that their integration can be achieved in a natural and principled fashion. A detailed architecture for integrated knowledge acquisition systems has been proposed that also derives from a parallel cognitive and theoretical studies.

An implementation of this architecture around a knowledge representation server has been described, and the server architecture has been detailed to the level of record structures and implementation experience.

As noted in the introduction, this paper presents a distillation of ideas from the whole community put forward as a contribution to a next generation of system designs. It now needs to be critically tested against other researchers' experiences and requirements.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. I am grateful to many colleagues for discussions over the years that have influenced the research described in this paper, in particular John Boose, Jeff Bradshaw, Bill Clancey, Debbie Leishman, Marc Linster, Alain Rappaport, Doug Skuce, Brian Woodward and other colleagues at the Knowledge Acquisition Workshops world-wide. I am particularly grateful to Mildred Shaw for research collaboration that has shaped many of the ideas reported in this paper.

REFERENCES

Bareiss, R. (1989). **Exemplar-Based Knowledge Acquisition**. New York: Academic Press.

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.

Borgida, A., Brachman, R.J., McGuiness, D.L. & Resnick, L.A. (1989). CLASSIC: a structural data model for objects. Clifford, J., Lindsay, B. & Maier, D., Eds. **Proceedings of 1989 ACM SIGMOD International Conference on the Management of Data.** pp.58-67. New York: ACM Press.

Brachman, R. J., Borgida, A., McGuinness, D. L., and Resnick, L. A. (1989). The CLASSIC knowledge representation system, or, KL-ONE: The next generation. **Workshop on Formal Aspects of Semantic Networks.** Santa Catalina Island, California (February).

Brachman, R. J., Gilbert, V.P. & Levesque, H. J. (1985). An essential hybrid reasoning system: knowledge and symbol level accounts of KRYPTON. **Proceedings of IJCAI85.** pp.547-551. Los Altos: Morgan Kaufmann.

Brachman, R. J., & Levesque, H. J. (1984). The tractability of subsumption in frame-based description languages. **Proceedings of AAAI-84.** pp.34-37. San Mateo, California: Morgan Kaufmann.

Brachman, R.J. & Schmolze, J. (1985). An overview of the KL-ONE knowledge representation system. **Cognitive Science,** 9(2) 171-216 (April-June).

Breuker, J. & Wielinga, B. (1987) Use of models in the interpretation of verbal data. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies** 27 (4), 349-370 (October).

Compton, P. & Jansen, R. (1989) A philosophical basis for knowledge acquisition. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89)** pp.75-89. Paris (July).

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.

Eshelman, L., Ehret, D., McDermott, J. & Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies, 26**(1), 41-54.

Gaines, B.R. (1988) Knowledge acquisition systems for rapid prototyping of expert systems. **INFOR, 26**(4), 256-285 (November).

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. (1987). Advanced expert system support environments. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of Second AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems.** pp.8-0-8-14. Banff (October).

Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. 47-66.

Gaines, B.R. & Shaw, M.L.G. (1989) Comparing the conceptual systems of experts. **Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.** pp.633-638. Los Angeles, California: Morgan Kaufmann (August).

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.9-1-9-24. Banff (November).

Ginsberg, M.L. (1987) **Readings in Non-Monotonic Reasoning.** San Mateo, California: Morgan Kaufmann.

Gomez, F. (1990) Knowledge acquisition from natural language for expert systems based on classification problem-solving methods. **Knowledge Acquisition 2** (2), to appear (June).

Graesser A.C. & Clark, L.F. (1985). **Structures and Procedures of Implicit Knowledge.** New Jersey: Ablex.

Hausser, R. (1987). **SComputation of Language.** Berlin: Springer.

Jones, W.P. (1989) Bringing corporate knowledge into focus with CAMEO. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.17-1-17-23. Banff (October).

Klinker, G., Bentolila, J., Genetet, S., Grimes, M. & McDermott, J. (1987) KNACK—report-driven knowledge acquisition. **International Journal of Man-Machine Studies, 26**(1), 65-79.

Leishman, D. (1989). Analogy as a constrained partial correspondence over conceptual graphs. **Proceedings of KR'89: First International Conference on Principles of Knowledge Representation and Reasoning.** pp.223-234. San Mateo, California: Morgan Kaufmann (May).

Levesque, H. J. (1984). A logic of implicit and explicit belief. **Proceedings of AAAI-84.** pp.198-202. San Mateo, California: Morgan Kaufmann.

MacGregor, R.M. (1988). A deductive pattern matcher. **Proceedings of AAAI88.** pp.403-408. San Mateo: Morgan Kaufmann.

Marcus, S. (1987). Taking backtracking with a grain of SALT. **International Journal of Man-Machine Studies, 26**(4), 383-398.

Michalski, R.S. & Chilausky, R.L. (1980). Knowledge acquisition by encoding expert rules versus computer induction from examples—A case study involving soyabean pathology. **International Journal of Man-Machine Studies 12,** 63-87.

Morik, K. (1987) Sloppy modeling. Morik, K. Ed. **Knowledge Representation and Organization in Machine Learning.** pp.107-134. Berlin: Springer-Verlag.

Monarch, I. & Nirenburg, S. (1987) The role of ontology in concept acquisition for knowledge-based systems. **Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'87).** Reading University, UK (September).

Motta, E., Eisenstadt, M., Pitman, K. & West, M. (1988) Support for knowledge acquisition in the knowledge engineer's assistant (KEATS). **Expert Systems 5**(1), 6-28.

Nebel, B. (1988). Computational complexity of terminological reasoning in BACK. **Artificial Intelligence, 34** 371-383.

Nebel, B. (1990) **Reasoning and Revision in Hybrid Representation Systems.** Berlin: Springer-Verlag.

Quinlan, J.R. (1987) Simplifying decision trees. **International Journal of Man-Machine Studies 27** (3), 221-234 (September).

Russell, S.J. (1989). **The Use of Knowledge in Analogy and Induction.** London: Pitman.

Schmidt-Schauss, M. (1989). Subsumption in KL-ONE is undecidable **Proceedings of KR'89: First International Conference on Principles of Knowledge Representation and Reasoning.** pp.421-431. San Mateo, California: Morgan Kaufmann (May).

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. & Woodward, J.B. (1989) Mental models in the knowledge acquisition process. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.29-1-29-24. Banff (October).

Skuce, D., Shenkang, W. & Beauville, Y. (1989) A generic Knowledge Acquisition Environment for conceptual and ontological analysis. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.31-1-31-20. Banff (October).

Watanabe, H. (1989). Heuristic graph displayer for G-BASE. **International Journal of Man-Machine Studies, 30**(3) 287-302 (March).

Wetter, T. & Woodward, J.B. (1989) Towards a theoretical framework for knowledge acquisition. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. submitted. Banff (November).

Woodward, B. (1990) Knowledge engineering at the front-end: defining the domain. **Knowledge Acquisition 2**(1), 73-94 (March).

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1990.

COGNITIVE AND LOGICAL FOUNDATIONS OF KNOWLEDGE ACQUISITION

Brian R Gaines & Mildred L G Shaw

Knowledge Science Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4
gaines@cpsc.ucalgary.ca & mildred@cpsc.ucalgary.ca

ABSTRACT

The widely used repertory grid knowledge elicitation methodology is based on *personal construct psychology* but utilizes primarily the notion of dichotomous constructs and does not encompass the ordinal relations between them captured in semantic net elicitation. This paper reviews Kelly's formal development of personal construct psychology and gives a systemic basis for it in terms of a calculus of distinctions. This basis is shown to be general enough to encompass category-theory and specific enough to define the computational structures of knowledge representation schema: concepts, constructs, individuals, roles, subsumption, disjoint concepts, primitive concepts and rules. This provides unified cognitive and logical foundations for knowledge acquisition that unifies current techniques and provides design principles for a new family of integrated tools.

INTRODUCTION

The repertory grid has become a widely used and accepted technique for knowledge elicitation, and has been implemented as a major component of many computer-based knowledge acquisition systems. The grid itself dates back to Kelly's (1955) application of *personal construct psychology* that he had developed on the basis of his clinical and teaching experience. The first computer-based elicitation and analysis system for repertory grids was developed by Shaw (1979) with applications mainly in educational, clinical and management studies. Gaines and Shaw (1980) suggested that repertory grids would provide a useful development technique for expert systems, and later published a validation study of the elicitation of the BIAIT methodology from accountants and accounting students using computer-based repertory grid elicitation and analysis (Shaw & Gaines, 1983). Boose (1984) in an independent parallel study reported success in a wide range of industrial knowledge-based system developments using computer elicitation of repertory grids, and since then many knowledge acquisition systems have incorporated repertory grids as a major elicitation technique (Boose & Bradshaw, 1987; Diederich, Ruhmann & May, 1987; Gaines & Shaw, 1987; Garg-Janardan & Salvendy, 1987; Shaw & Gaines, 1987; Ford, Stahl, Adams-Webber, Novak & Jones, 1990).

The 'repertory grid' methodology has itself evolved in the light of application experience and now has major differences from that described by Kelly. Shaw (1980) took advantage of the processing power and interactivity of computers to introduce on-line analysis and feedback to the person from whom the grid was being elicited. In knowledge-based systems terms, this can be seen as highlighting correlations that might be spurious and lead to incorrect rules in later analysis. Shaw and Gaines (1979) introduced new forms of analysis of the repertory grid based on fuzzy sets theory which became the basis of rule extraction (Gaines & Shaw, 1986). Boose and Bradshaw (1987) made changes to the grid structure introducing hierarchical data structures to cope with more complex domains. Bradshaw, Boose, Covington and Russo (1988) showed how many problems that did not seem appropriate to repertory grids could be formulated in terms of them.

The repertory grid was an instrument designed by Kelly to bypass cognitive defences and give access to a person's underlying construction system by asking them to compare and contrast relevant examples (significant people in their life in the original application). This has also been its significance in expertise transfer, that it is often easier and more accurate for the expert to provide critical cases rather than a domain ontology. The power of a few critical cases described in terms of relevant attributes to build a domain ontology is remarkable but statistically well-founded (Gaines, 1989), and is the basis of the success of repertory-grid techniques.

However, the repertory grid itself is based primarily on only one aspect of Kelly's personal construct psychology, his *dichotomy corollary* that, "A person's construction system is composed of a finite number of dichotomous constructs." The standard grid is a flat structure of elements described in terms of dichotomous attributes or constructs that does not represent the hierarchical structure of Kelly's organization corollary that, "Each person characteristically evolves, for his convenience of anticipating events, a construction system embracing ordinal relationships between constructs." Hinkle (1965) developed a technique of *laddering*, based on "why" and "how" questions, for investigating ordinal relations between constructs, and Boose (1986) incorporated a laddering tool in ETS, but ordinal relations between constructs have not been the primary focus in applications of repertory grid tools.

Ordinal relations are a primary focus in tools that use some form of semantic network to build an overall domain ontology directly, such as MOLE (Eshelman, Ehret., McDermott & Tan, 1987), KNACK (Klinker, Bentolila, Genetet, Grimes, & McDermott, 1987), SALT (Marcus, 1987), ONTOS (Monarch & Nirenburg, 1987), Cognosys (Woodward, 1990), KEATS (Motta, Eisenstadt, Pitman & West, 1988), CODE (Skuce, Shenkang & Beauville, 1989), CAMEO (Jones, 1989), and in techniques based on domain ontologies such as KADS (Breuker & Wielinga, 1987). The tools vary in their interfaces, sources, emphasis on declarative or procedural knowledge, and domain dependence, but all result in domain models based on a network of directed relations between concepts. These tools are different in approach from those based on repertory grids, but as the trend towards the integration of tools and techniques continues it is becoming important to understand the relationship between them. Similar considerations apply to other tools now being incorporated in integrated knowledge acquisition systems such as those for empirical induction and conceptual clustering.

This paper develops cognitive foundations for current knowledge acquisition tools and techniques in terms of Kelly's personal construct psychology. It uses a systemic analysis of the psychology in terms of a calculus of distinctions (Gaines & Shaw, 1981) to provide a framework for knowledge acquisition that is both psychologically and logically well-founded, and which has direct, operational interpretation in terms of knowledge acquisition tools and techniques.

PERSONAL CONSTRUCT PSYCHOLOGY

Kelly (1955) put forward the idea of an individual using "personal constructs" as filters through which he perceives events: "Man looks at his world through transparent templets which he creates and then attempts to fit over the realities of which the world is composed." (pp.8-9) He continually emphasizes the epistemological status of these constructs in predicting and controlling the world and their ontological status as personal conjectures rather than reality-derived absolutes: "Constructs are used for predictions of things to come, and the world keeps on rolling on and revealing these predictions to be either correct or misleading. This fact provides the basis for the revision of constructs and, eventually, of whole construct systems." (p.14)

Kelly was acutely aware of the psychologist as an observer modeling a person and objected to the introduction of theoretical constructs such as "learning" and "motivation" as if they were causal variables. Hence, he was extremely careful in the exposition of his own system to make its underlying presuppositions minimal and overt by developing personal construct psychology as a semi-formal system based on a postulate and corollaries. He made minor but significant changes to the corollaries after the publication of his book, and those below are updated from Bannister and Mair (1968).

"**Fundamental Postulate:** A person's processes are psychologically channelized by the way in which he anticipates events."

This was stated as a postulate to emphasize that it was presented as a convenient viewpoint from which to understand human behavior, not imputed to an underlying physiological or psychological reality. This is consistent with modern views of the role of theory in science, but was a fairly radical position in Kelly's time. The basis of Kelly's approach to psychotherapy was that this was also a convenient viewpoint from which someone could understand, and modify, their own behavior. He saw all people as "personal scientists" in anticipating the world, and attempted to develop techniques where this anticipatory modeling activity was reflexively applied to the self.

This fundamental emphasis on the role in behavior of a view to the future is what distinguishes Kelly's approach to psychology. He saw man as driven by the need to cope with coming events in the world and all other aspects of behavior as deriving from this. He notes in expanding upon this point that "a person's processes, psychologically speaking, slip into the grooves which are cut out by the mechanisms he adopts for realizing his objectives." (Kelly, 1955, p.49)

"**Construction Corollary**: A person anticipates events by construing their replications."

Construction involves two basic operations, slicing out an "event" and interpreting it by "construing." Kelly notes "we look at the undifferentiated stream of circumstance flowing past us, and we try to find something about it that repeats itself. Once we have abstracted that property, we have a basis for slicing off chunks of time and reality and holding them up for inspection one at a time...this is done by construing" (p.120) "By construing we mean placing an interpretation: a person places an interpretation on what is construed. He erects a structure, within the framework of which the substance takes shape or assumes meaning." (p.50). "When one abstracts replicated properties in the events he has already experienced, it becomes possible for him to chart events to come in terms of these same properties. ... What one predicts is not a fully fleshed-out event, but simply the common intersect of a set of properties. If an event comes along in which all the properties intersect in the prescribed way, one identifies it as the event he expected." (p.121)

It is these "slices of circumstance" that we elicit from an expert as "critical cases", and these "interpretations" that we elicit as "relevant attributes."

"**Individuality Corollary**: Persons differ from each other in their constructions of events."

Kelly stresses the personal nature of construing but he emphasizes also that "persons can find common ground through construing the experiences of their neighbors along with their own", although "individuals can be found living out their existence next door to each other but in altogether different subjective worlds." (p.56) In knowledge-based system terms it is the personal, non-overt, nature of expertise that makes knowledge elicitation from experts a necessary developmental technique.

"**Organization Corollary**: Each person characteristically evolves, for his convenience of anticipating events, a construction system embracing ordinal relationships between constructs."

Kelly notes that "man systematizes his constructs by concretely arranging them in hierarchies and by abstracting them further" (p.58) He saw the hierarchy as one in which the stability of the upper levels allowed for change in the lower ones and that the arrangement of constructs in a particular hierarchy was more a determinant of personality than the constructs themselves. In knowledge-based system terms, hierarchies and abstractions are the material of semantic nets and software engineering. Kelly's approach emphasizes that they are constructs of the modeling system and characterize its particular "expertise."

"**Dichotomy Corollary**: A person's construction system is composed of a finite number of dichotomous constructs."

Kelly's emphasis on the dichotomous, bipolar nature of constructs is what most distinguishes them from what are conventionally called "concepts." "The construct, of itself, is the kind of contrast one perceives... while constructs do not represent or symbolize events, they do enable us to cope with events, which is a statement of a quite different order...They also enable us to put events into arrays or scales, if we wish." (pp.13-14) Kelly saw the linked concepts of *similarity* and *contrast* between elements forming the two "poles" of a construct. This does not necessarily involve a binary evaluation of elements since they may lie somewhere between the poles.

Kelly emphasizes that the opposition between the poles of a construct is not that of logical negation—"Now conventional logic would say that black and white should be treated as separate concepts. Moreover, it would say that the opposite of black can only be stated as not black, and the opposite of white can only be stated as not white. Thus the person whose field we mentioned would have shoes which would be just as much not white as the time of day, and he would write on paper which would be just as not black as the distance to his office." (p.106)

The notion that dichotomous pairs of concepts are fundamental to human mental process is an ancient one, strongly present in Greek philosophy (Lloyd, 1966), and found in many anthropological studies of the concepts defining a culture (Maybury-Lewis & Almagor, 1989). Eliciting dichotomous constructs by asking in what way are two entities in a domain alike and different from a third is the basic technique used in repertory grids. Semantic net elicitation focuses on unipolar concepts, although the meta-concept that two concepts are *disjoint* from one another can be significant.

The relation between constructs and concepts is both psychologically and theoretically important, and the next section analyzes it in detail.

"**Choice Corollary**: A person chooses for himself that alternative in a dichotomized construct through which he anticipates the greater possibility for the elaboration of his system."

This is Kelly's basis for human action which he saw as an alternative to concepts of motivation—"there is a continuing movement toward the anticipation of events, rather than a series of barters for temporal satisfactions" (p.68).

"**Range Corollary**: A construct is convenient for the anticipation of a finite range of events only."

This may be thought of as introducing a concept of relevance into the definition of construct systems.

"**Experience Corollary**: A person's construction system varies as he successively construes the replications of events."

This expresses the essential fallibility and conjectural nature of constructs. They are hypotheses to be judged by their utility and updated as they fail in anticipation. This updating is what the observer construes as "learning."

"**Modulation Corollary**: The variation in a person's construction system is limited by the permeability of the constructs within whose ranges of convenience the variants lie."

Kelly sees a "permeable" construct as one "so constituted that new experience and new events can be discriminatively added to those which it already embraces." (p.81)

"**Fragmentation Corollary**: A person may successively employ a variety of construction subsystems which are inferentially incompatible with each other."

Through this corollary Kelly allows for the operations of different psychological systems and hence different methods of inference and forms of rationality within a single individual. Pask (1975) sees "roles" within a person in similar terms, as forming separate psychological individuals, "P-Individuals." In these terms, an "expert" is not a person but a specialized and valued role within a person.

"**Commonality Corollary**: To the extent that one person embraces a construction of experience which is similar to that employed by another, his processes are psychologically similar to those of the other person."

This is complementary to the individuality corollary and allows for the "fact that certain groups of people behave similarly in certain respects." (p.93)

"**Sociality Corollary**: To the extent that one person construes the construction processes of another, he may play a role in a social process involving the other person."

Kelly is here allowing the construal processes of one person to become elements for construal by another and hence to model the process by which we anticipate the actions of others.

MAKING DISTINCTIONS

A theoretical framework that encompasses personal construct psychology, knowledge representation and acquisition for knowledge-based systems, and the relations between these topics, can be based on a single fundamental primitive, what Brown in his *Laws of Form* terms "making a distinction."

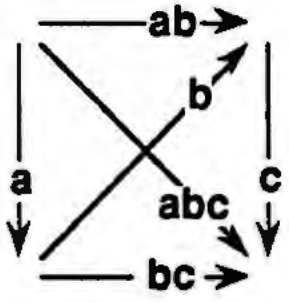
> "The theme of this book is that a universe comes into being when a space is severed or taken apart... By tracing the way we represent such a severance, we can begin to reconstruct, with an accuracy and coverage that appear almost uncanny, the basic forms underlying linguistic, mathematical, physical and biological science, and can begin to see how the familiar laws of our own experience follow inexorably from the original act of severance." (Brown, 1969 p.v)

We have shown that a wide range of psychological and philosophical frameworks are characterized by the systems of distinctions underlying them, and that there are a number of basic distinctions common to such systems (Gaines & Shaw, 1984). The frameworks included that of Kelly's personal construct psychology, and this section defines those parts of a calculus of distinctions adequate to do this.

Formally, a *distinction* is taken to be a primitive notion, but its connotations of cutting out something for some purpose, or because it satisfies a criterion, are informally intended. The connotations that there may be an agent making the distinction, and that something is distinguished are also informally useful. However, it should be noted that closed-world assumptions are specifically excluded, it is not assumed that what is *not* distinguished is meaningful or has the same categorial status as that which is distinguished. Informally, only that which is distinguished is known, new distinctions can be made indefinitely, and the world is assumed open to novelty. Formally, this affects the way relations between distinctions are defined. Extensional forms, such as "for all", are avoided because it is not reasonable to assume that statements can be made about an indefinite extension whose members are not yet identified and may never be known.

Distinction are the only primitives. One might ask "what is distinguished?", but the answer will be "distinctions." This corresponds to deriving the two phenomena in Kelly's *construction corollary* from a single primitive, that we distinguish events in the undifferentiated stream of circumstance, and then we further distinguish among the distinguished events by construing.

Translating this into an algebraic formalism, if distinctions are represented by bold lower case letters, and a distinction applied to another distinction is concatenated to the right of it, what has been defined is a partial groupoid (Bruck, 1971) over an alphabet of distinctions, that is a set of expressions of the form: **a, b, c, ad, bd, be, ce, cf,** and so on. Allowing further distinctions to be applied to those already distinguished, so that expressions of the form **adg** have meaning, extends the groupoid. If, in addition, sequences of distinctions can be named as a single distinction, with the naming convention that these are to be interpreted only through context-insensitive expansion, the groupoid becomes a semigroupoid. That is, it is associative: **a(bc)** = **(ab)c** where the equality may be interpreted as "means the same as." In graphical terms, the following diagram commutes:



This semigroupoid structure turns out to be adequate for exposition of personal construct psychology, and for current knowledge representation schema. This rich capability from such slight resources is, perhaps, less surprising if it is noted that adding the requirement for left and right identity distinctions to the above structure gives the formal structure of a *category* (MacLane, 1971), an algebraic structure sufficient to support all current systems of mathematics and logic (MacLane, 1986). Thus, the interesting questions are about the extent and naturality of the infrastructure that one has to erect in terms of the basic calculus of distinctions in order to derive an adequate semantics for personal construct psychology and knowledge representation.

DISTINCTIONS AMONG DISTINCTIONS

One distinction will be said to *subsume* another if it can always be applied whenever the other can. This notion of subsumption turns out to be adequate to capture Kelly's use of the term that one construct subsumes another, and also the use of the same term in knowledge representation, that one concept subsumes another. It can be represented formally as:

"**b** subsumes **a**" $\quad \mathbf{a} \rightarrow \mathbf{b} \Leftrightarrow \vdash \mathbf{xa} \Rightarrow \vdash \mathbf{xb}$ (1)

That is, **b** subsumes **a**, if and only if whenever one asserts **xa** one also asserts **xb**. Note that the form $\forall \mathbf{x}$ is avoided for the reasons given above, that the notion of all the distinctions to which **a** and **b** may be applied is not well-defined. However, if this extension is well-defined then the intensional formulation of (1) implies the extensional formulation in terms of $\forall \mathbf{x}$. Marcus' (1962) non-standard, *substitutional* reading of the universal quantifier would be appropriate to the definition above, but the notation is avoided because the extensional reading is prevalent and significant in its own right.

The definition in (1) is best read intensionally in terms of a *commitment* to the way in which distinctions will be made, such that if **a** is made then there is a commitment to **b** being made also. Subsumption corresponds to increasing generality since the subsuming distinction can be applied to at least as many things as that subsumed. It is an asymmetric, transitive relation, a partial pre-order, over distinctions, that supports the ordinal relations of Kelly's *organization corollary*.

There is a second relation definable in similar terms which is equally important, that one distinction is *disjoint* with another in that one can never be applied whenever the other can. This definition of a disjoint relation turns out to be adequate to capture Kelly's notion of a dichotomy, and also the notion of disjoint concepts in knowledge representation. It can be represented formally as:

"**a** disjoint **b**" $\quad \mathbf{a—b} \Leftrightarrow \vdash \mathbf{xa} \Rightarrow \neg \vdash \mathbf{xb}$ (2)

That is, **a** is disjoint with **b**, if and only if whenever one asserts **xa** one does not assert **xb**. The definition is again best read intensionally in terms of a commitment to the way in which distinctions will be made, such that if **a** is made then there is a commitment to **b** not being made. Disjoint is a symmetric, intransitive relation over distinctions, and supports Kelly's *dichotomy corollary*.

The $\rightarrow$ and — relations are complementary in establishing four possible binary relations between distinctions, that $\mathbf{a \rightarrow b}$, $\mathbf{b \rightarrow a}$, **a—b**, or none of these. The two subsumption relations can hold together giving an equivalence relation on distinctions. The disjoint relation is incompatible with the subsumption relations, and is *inherited* through subsumption, that is:

$$\mathbf{a—b} \text{ and } \mathbf{a \rightarrow c} \Rightarrow \mathbf{c—b} \qquad (3)$$

These notions are sufficient for the definition of concepts and constructs. A *concept* is defined to be *that mental entity imputed to a distinction making agent as enabling it to make a particular distinction.*

Note that concepts are separated both from the distinctions they support and the entities they distinguish. This definition corresponds to Anglin's (1977, p.3), "a concept is all of the knowledge possessed by an individual about a category of objects or events", "Concepts mediate categorization but concepts are not the resultant categories." The notion of an agent and its mentality are deliberately introduced at this stage because they are part of the connotations of a concept. However, consistent with Kelly's position, concepts are defined to be *imputed* to an agent rather than as a mental structure. They are themselves distinctions made by an observer, possibly a reflective observer. This is equivalent to Zadeh's (1964) introduction of the notion of the *state* of a system as an imputed parametrization accounting for a system's behavior. *Concepts are state variables we impute to a knowledgeable agent.*

A construct is defined formally to be a triple of two disjoint distinctions mutually subsumed by a third as shown in Figure 1, and psychologically as the triple of concepts assumed to underlie the distinctions.

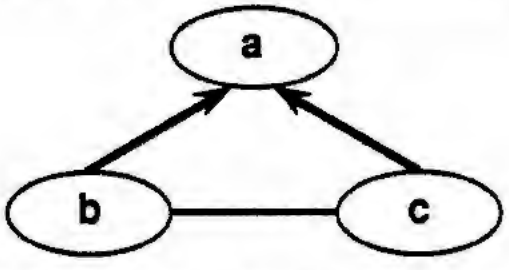


Fig.1 The triple of distinctions generating a construct

The triple captures the notions of *similarity* through the shared subsuming concept, **a**, and *contrast* through the disjoint subsumed concepts, **b** and **c**. The pair of disjoint concepts, **b** and **c**, captures Kelly's *dichotomy corollary*, that a person's construction system is composed of a finite number of dichotomous constructs. The concepts **b** and **c** characterize what Kelly terms the *poles* of the construct. A minimal subsuming concept captures Kelly's *range corollary*, that a construct is convenient for the anticipation of a finite range of events only. Such a minimal concept **a** characterizes what Kelly terms the *range of convenience* of the construct. The importance of such triples derives from the restricted closed world semantics generated, that the world carved out by concept **a** is divided by the disjoint concepts **b** and **c** into two parts that are now at the same categorial level.

The construction of Figure 1 can be simplified by noting that a subsuming concept **a** can be constructed for a pair of disjoint concepts **b** and **c**, and it is the disjunction which primarily defines the construct. In the graph of the disjoint relation between concepts, concepts are nodes and constructs are arcs. Hence *concept and construct are graph-theoretically dual relations.*

CONCEPTUAL STRUCTURES

The subsumes and disjoint relations, → and —, can between them characterize a wide variety of concept and construct structures. The left hand structure in Figure 2 shows that a given range of convenience can be construed through a number of unrelated constructs. The pairs **b**—**c** and **d**—**e** are two different dimensions of the domain characterized by **a**. If **a** characterizes a domain of people then **b**—**c** might be **young**—**old** and **d**—**e** might be **short**—**tall.**

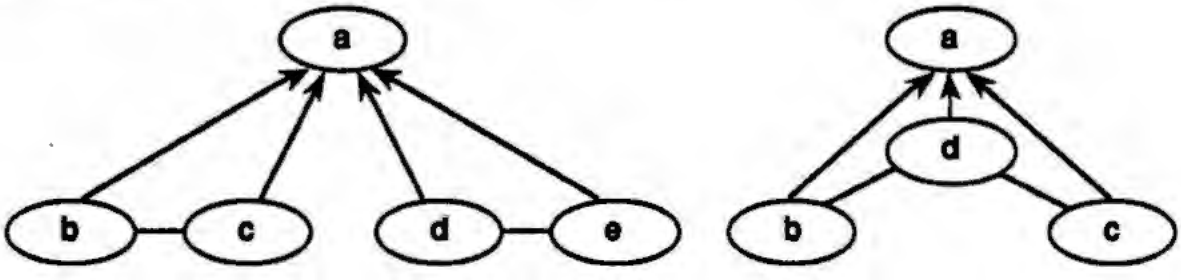


Fig.2 Alternative constructs and alternative poles

The right hand structure in Figure 2 shows that a pole of a construct may be paired with alternative poles that are not mutually disjoint. The pairs **b**—**d** and **c**—**d** are two different dimensions of the domain characterized by **a** that have some commonality through the shared pole, **d**. If **a** characterizes a domain of people then **b**—**d** might be **young**—**old** and **c**—**d** might be **fun**—**old**. It was because different contrasts were possible for the same concept that Kelly saw the concept pairs that form constructs as being a more stable chararacterization of psychological structure, that the way in which we react to something falling under a concept is dependent on what we see as the contrasting concept.

If the concepts **b** and **c** on the right of Figure 2 are also disjoint as shown on the left of Figure 3 then the structure may either be regarded as a trichotomy or a triple of dichotomous constructs. In knowledge representation terms, **b**, **c** and **d** might be seen as alternative values along some dimension such as young, old and middle-aged respectively. In personal construct terms one can see that the contrasts **young**—**middle-aged, middle-aged**—**old,** and **young**—**old,** might all be used in different situations.

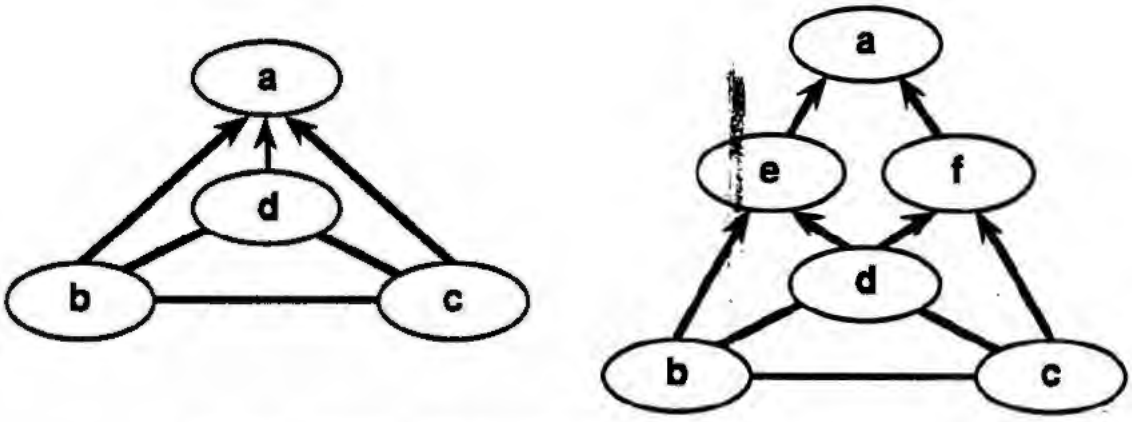


Fig.3 Trichotomous construct and a scalar trichotomy

What the structure on the left of Figure 3 does not capture is the possibility of putting events along scales which Kelly mentions in his dichotomy corollary. Middle-aged is not just an alternative to young and old, but an intermediary possibility between them. The structure on the right of Figure 3 extends that on the left to show how scales may be developed. The concepts **e** and **f** are not disjoint and **d** inherits their commonality while **b**, since it is disjoint with **d**, inherits a different part of **e**, and **c** similarly inherits a different part of **f**. For example, suppose that **b** is young interpreted as an age range [0, 35], **c** is old as an age range [50, 100], **d** middle-aged [35, 50], then **e** might be the age range [0, 50] and **f** [35, 100].

It is possible for the pair **e f** to be used as if it were a dichotomous construct even though the concepts are not disjoint. This generates the underlying infrastructure shown in Figure 4 where **d** captures the borderline cases (Sanford, 1975) that come under both **e** and **f**, **b**—**c** is a dichotomous construct underlying **e**—**f**, and **g**—**h** is a dichotomous construct characterizing that part of the borderline under **b** and that under **c**. The structure **b**, **c**, **d** is the basis for a fuzzy sets (Zadeh, 1965) characterization of the pseudo-construct **e f**, with, for example: **b** characterizing a membership of 1 to **e** and 0 to **f**; **c** a membership of 0 to **e** and 1 to **f**; and **d** 0.5 to **e** and 0.5 to **f**. Distinguishing more concepts subsumed by **d** allows for finer grades of membership (Gaines, 1976).

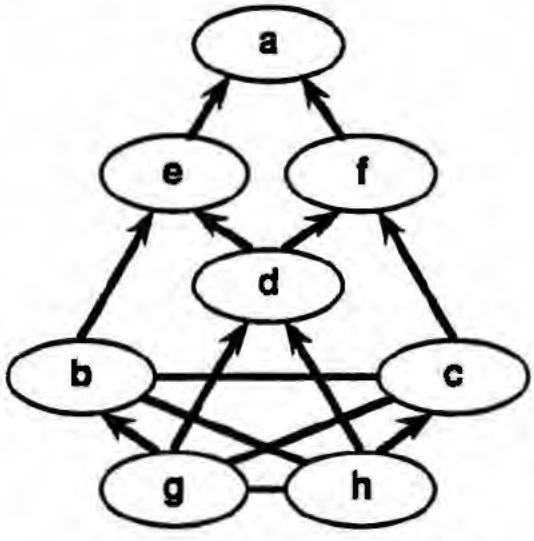


Fig.4 A non-dichotomous pseudo-construct, borderline and fuzzy infrastructure

The relation of **g—h** to **b—c** in Figure 4 is an example of an ordinal relation between constructs as described by Kelly in his *organization corollary*. Figure 5 shows two such ordinal relations discussed by Kelly. On the left, the concept characterized by the triple **d, e, f,** is subordinate to that characterized by **a, b, c.** For example, **b—c** might be **good—bad**, and **e—f** might be **fun—boring.** When the upper construct is evaluatory, such as **good—bad**, its subordinate constructs inherit the evaluatory connotations and typically have a *preferred pole.*

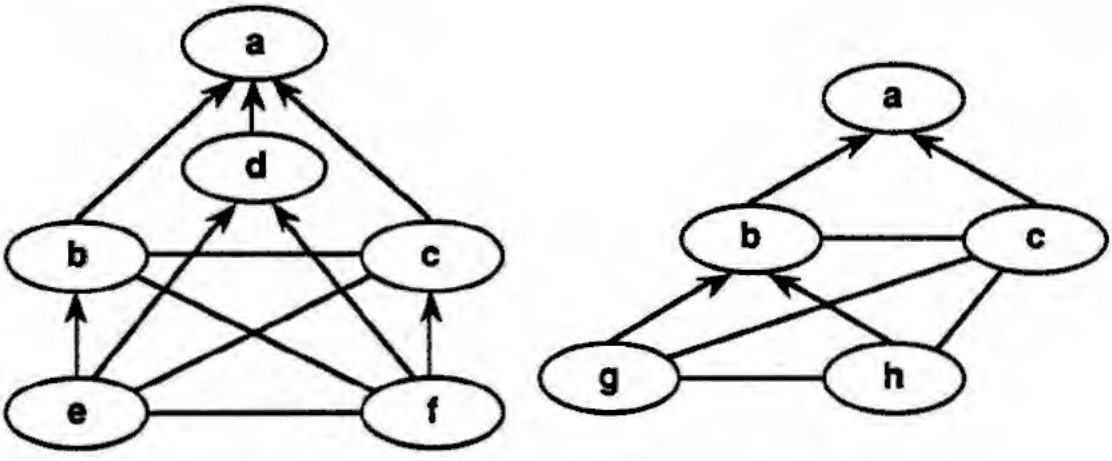


Fig.5 Some ordinal relations between constructs

On the right of Figure 5, the concept characterized by the triple **b, g, h,** is subordinate to that characterized by **a, b, c.** For example, **b—c** might be **good—bad**, and **g—h** might be **dancing—swimming.** The ordinal relations between constructs derive from the ordinal relation of subsumption between the underlying distinctions and an indefinite number of subordination structures are possible.

KNOWLEDGE REPRESENTATION

The calculus of distinctions developed above and discussed in terms of its cognitive connotations in personal construct psychology also provides computational foundations for knowledge representation schema such as those of KL-ONE (Brachman & Schmolze, 1985) and CLASSIC (Borgida, Brachman, McGuinness & Resnick, 1989). These schema involve concepts, primitive concepts, disjoint concepts, individuals, roles, constraints upon roles, individuals filling roles, rules, subsumption between concepts, and recognition of individuals by concepts. They encompass semantic nets, frames, and rule-based systems, as well as relational and post-relational databases. This section shows how such systems may be derived from primitives underlying systems of distinctions.

A *concept* in a computational context is defined to be *that data structure used by a distinction making algorithm to enable it to make a particular distinction.* No imputation is necessary because we are not dealing with identification of a black box from its behavior, but rather with the internal structure of an accessible white box. The *disjoint* relation between concepts already defined corresponds precisely to the disjoint relation in knowledge representation schema. Similarly, the *subsumption* relation between concepts already defined corresponds precisely to the "is-a" subsumption relation in knowledge representation schema. The interpretation of subsumption in terms of commitment above corresponds to the definitional form of the "is-a" relation. The computed form of "is-a" requires some further structures which are developed below, and also allow primitive and non-primitive concepts to be defined.

Individuals in a computational context are defined to be *those distinctions which are initial in that they cannot be preceded by other distinctions*. This corresponds to the notion of an *initial object* in category theory and also to the analysis of individuals in metaphysics. Gracia (1983) notes that traditionally individuals are characterized by five properties; they are entities which:
(a) lose their fundamental character if they are divided into parts;
(b) are distinct from all other entities, even those that are of their same type or kind;
(c) are part of a group type or class which has or can have several members;
(d) remain fundamentally the same through time and various changes;
(e) are the subject of predication but not predicable of other things.
Gracia argues against each of these in turn as being a necessary or sufficient feature to characterize individuals and suggests instead that it is *non-instantiability* that characterizes an individual; that is we cannot instantiate an individual. This corresponds to an initial distinction which cannot have another, an instance, to its left.

Thus, while a distinction is the sole primitive, the difference between initial distinctions and non-initial ones is an important one. Individuals are concepts that cannot be applied to other concepts and hence have no extensions or instances and cannot subsume one another. However, since individuals are concepts it is possible to have knowledge representation schemas which do not make the distinction explicitly. It is also possible to treat non-initial concepts as *abstract individuals* that represent all the individuals that might be distinguished by them. When such an abstract individual has no extension, that is, there are not, or cannot be, any individuals that it distinguishes, then the paradoxes of sense and reference studied by Frege (1892), Meinong (1910) and Russell (1905), can arise. The "morning star, evening star" and "Scott, author of Waverley" equivalences are not problematic as they corresponding to two concepts subsuming the same individual. Non-existent objects such as "the present King of France" are also not problematic because there is no dependence on their empty extensions. They are interesting because, like fictional objects (Woods, 1974), they mimic individuals in that they do not (rather than cannot) have another distinction to their left. This may well underlie Meinong's intuition that they are, in some sense, equivalent to real objects.

A *role, property, attribute* or *relation* arises naturally by an important construction as shown in Figure 6. The triple of concepts defining a construct on the upper left may be used to generate that on the right by *defining* the concept **d** to be precisely the union of concepts **b** and **c**, in the sense that **d** subsumes a distinction if, and only if, it is subsumed by **b** or subsumed by **c**:

"**d** internalizes **b**—**c**" $\quad$ **d**←**b**—**c** $\Leftrightarrow \vdash \mathbf{xd} \Leftrightarrow \vdash \mathbf{xb} \vee \vdash \mathbf{xc}$ (4)

The definition in (4) reads as a commitment to making the distinction **d** if and only if there is a commitment to either **b** or **c** being made also. As shown in the lower left of Figure 6, the concept **d** is uniquely minimal in the subsumption ordering as a range of convenience for the construct defined by **a** , **b**, and **c**. Thus, the construct defined by **d**, **b**, and **c** precisely and uniquely captures the essence of the original construct.

The notation shown in the definition of **d** indicates that it is defined as subsuming the construct **b**—**c** and subsumes what is distinguished by **b** or by **c**. The construct **b**—**c** becomes a *role* and the alternation of the concepts **b** or **c** becomes a *constraint* on the role. This internalization process can be applied to other structures also such as the trichotomies of Figure 3 in which case the role can take more than two values, as a categorical role (left of Figure 3) or scale (right of Figure 3). Non-dichotomous pseudo-constructs, such as that of Figure 4, may also be internalized giving multiple-valued roles or relations. It is natural also, particularly as more complex structures are internalized, to name the role in its own right rather than through its poles.

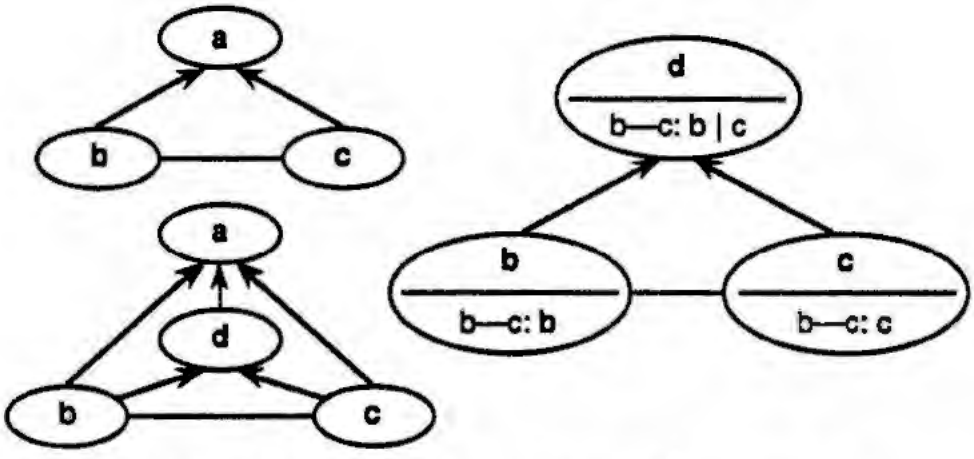


Fig.6 A role as the internalization of a construct

Multiple constructs with the same range of convenience may be internalized to define a concept with multiple roles as shown in Figure 7. The concept i may not be precisely superordinate to the constructs **b—c** and **d—e** because they may not cover the same range, but it does arise as the subordination of **g** and **h** which are the precise forms of **b—c** and **d—e**.

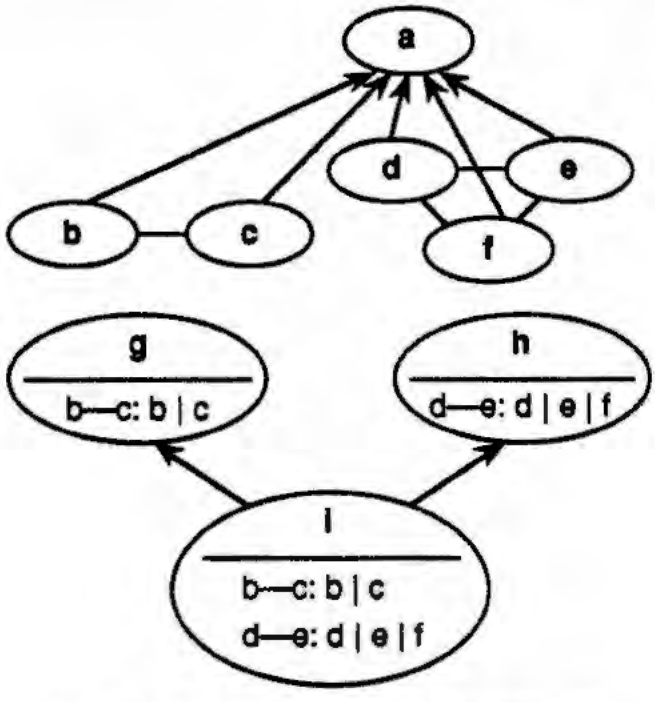


Fig.7 Multiple roles defining a concept

The internalization of constructs to form roles and constraints provides a natural coordinate system specifying individuals in terms of the concepts that distinguish them. The assertion of **xb** may be represented as the individual **x** having the value **b** on the role **b—c**. Note that, as individuals cannot subsume concepts, it is not possible to use the internalization construction. The coordinate system appears similar but is generated through a different construction. This corresponds to Zalta's (1983) distinction in his intensional logic of abstract objects that *abstract objects **encode** properties whereas existent objects cannot encode properties but do **exhibit** them.*

The internalization construction of Figures 6 and 7 gives some concepts a defined internal structure in which constructs define coordinate axes named as roles and constrained by the concepts forming the poles of the construct. Individuals have values on these axes and the values are themselves concepts or individuals. There is an important distinction between concepts that results from this construction: the subsumption relation between concepts defined completely in terms of their internal structure is determined by that structure rather than by a separate commitment. Such structurally defined concepts are said to be *non-primitive* and the others *primitive*.

Thus the "is-a" relation for non-primitive concepts is computable rather than definable, the commitment to their definition in terms of their structure entails a commitment to a computed, rather than a defined, "is-a" relation. Confusion about these two forms of concept, and associated "is-a" relations, caused problems in early developments of semantic nets (Brachman, 1983). A similar confusion was also a problem in differentiating the definition of concepts from the assertion of those applicable to individuals (Woods, 1975), the encoding—exhibiting distinction above. The generation of concepts, roles, individuals, constraints and values from the single primitive of making a distinction clarifies all these issues.

An important consequence of the structural definition of non-primitive concepts is that individuals may be *recognized* as being distinguished by one of these concepts even if this distinction has not been asserted. That is, other assertions made about an individual may be sufficient for them to satisfy the constraints of a non-primitive concept. This enables the anticipatory origins of distinction-making specified in Kelly's fundamental postulate to be captured. A *rule* is defined to be a link between concepts such that the recognition of an individual as distinguished by the first concept leads to the assertion that it is distinguished by the second:

"a rule b" $\quad \mathbf{a} \rightarrow\rightarrow \mathbf{b} \Leftrightarrow \mathbf{xa} \Leftrightarrow \vdash \mathbf{xb}$ (5)

Notice that this differs from subsumption as defined in (1) in that **xa** need not be asserted, the commitment is to assert **xb** if **xa** is noted through assertion or recognition. The double arrow notation is chosen to portray both the similarities and differences between a rule and a subsumption relation.

Rules allow a cognitive system to be anticipatory in that it contains structures which from one set of distinctions made about an event will infer that another should be made. This can lead to prediction or action. Rules play a similar role in computational systems in allowing recommendations for decision or action to be made. Overtly modeling the conceptual system of an expert as a structure of concepts, roles, individuals and rules is a basis for emulating the expert's performance in a knowledge-based system.

DISTANCE MEASURES, CONCEPTUAL CLUSTERING AND INDUCTION

Distance measures play an important role in conceptual clustering and induction. There is a natural construction, a distance between two concepts, **x** and **y**, as shown on the left of Figure 8. Let **u** be some minimal upper bound of **x** and **y** subsuming both of them, and **l** some maximal lower bound subsumed by both of them, and **U** be the extension of **u** and **L** the extension of **l** over some universe of individuals. If **x** and **y** are identical so will be **U** and **L**, whereas if they are disjoint **L** will be empty. Hence a natural distance measure is the number of individuals that are in **U** but not **L**:

"x distance y" $\quad d(\mathbf{a}, \mathbf{b}) = \mathcal{C}\mathbf{U} - \mathcal{C}\mathbf{L}$ (6)

where $\mathcal{C}\mathbf{U}$ and $\mathcal{C}\mathbf{L}$ are the cardinalities of **U** and **L** respectively. This measure satisfies the triangle inequality and can be normalized by dividing by its maximum possible value, $\mathcal{C}\mathbf{U}$. It is clearly dependent on the universe of individuals involved, but this is appropriate to measuring concept distance in an extensional context. Intensional concept "distance" independent of context is reflected in the relational structures already developed.

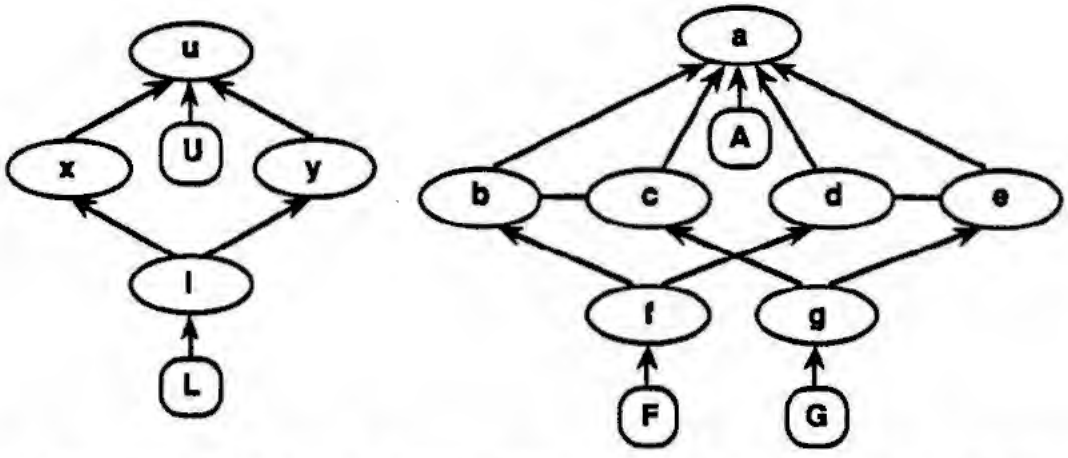


Fig.8 Calculation of distance measures between concepts and between constructs

The distance measure defined readily extends to constructs through the comparison of poles as shown on the right of Figure 8:

"b—c distance d—e" $\quad d(b—c, d—e) = \mathcal{C}A - \mathcal{C}F - \mathcal{C}G \qquad (7)$

This measure is a count of the numbers of individuals that fall under the opposite pole of the other construct. Note that it is not invariant if one construct is reversed.

What of trichtomous constructs such as those shown in Figure 3? The construction of Figure 8 readily extends to the categorical case on the left but does not capture the scalar semantics on the right. The construction on the left of figure 9 shows how the intermediate poles may be taken into account by also examining distances between the upper concepts that represent the scale:

$$d(b—d—c, g—l—h) = (\mathcal{C}A - \mathcal{C}F - \mathcal{C}G - \mathcal{C}G) + (\mathcal{C}A - \mathcal{C}O - \mathcal{C}P) \qquad (8)$$

This measure counts one for an individual at the extreme of one construct that falls under the center pole of the other, and two if it under the opposite extreme. This construction generalizes to scales with more than three points. If these scales are numbered linearly it computes a "city block" distance measure, which is precisely that used in construct clustering algorithms such as FOCUS (Shaw, 1980).

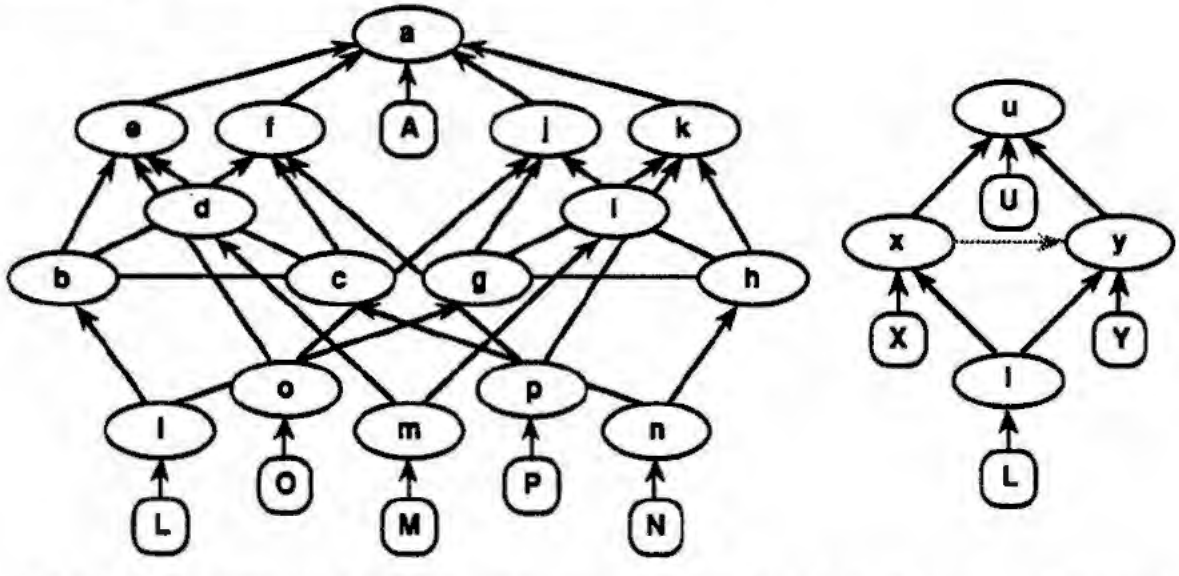


Fig.9 Scalar measures between constructs and induction of rules between concepts

These distance measures when applied to the constructs of one person enable natural clusters to be seen that may be grouped as part of a coherent concept, fo example, in that they are all contributors to an evaluatory dimension. When applied to the constructs of different people in the same domain they enable consensus, conflict, correspondence and contrast in the use of constructs to be measured (Shaw & Gaines, 1989).

The measures used in the induction of a rule linking to concepts are also readily derived as shown on the right of Figure 9. $\mathcal{CX}$ is the number of "predictions" made by concept x as the left hand side of a rule, and $\mathcal{CL}$ is the number which are correct. Thus, the measures of the validity of inducing the rule, $x \rightarrow y$, are:

"prior probability of y" $p(y) = \mathcal{CY}/\mathcal{CU}$ (9)

"probability correct $x \rightarrow\rightarrow y$" $p(x \rightarrow\rightarrow y) = \mathcal{CL}/\mathcal{CX}$ (10)

"probability by chance $x \rightarrow\rightarrow y$" $c(x \rightarrow\rightarrow y) = I_{p(y)}(\mathcal{CX}-\mathcal{CL}, \mathcal{CL}+1)$ (11)

where I is the incomplete beta function summing a binomial distribution tail. These measures are precisely those used by Induct (Gaines, 1989) inducing rules from datasets.

PRACTICAL APPLICATIONS

The knowledge representation primitives developed in this paper are not just theoretical constructs but have immediate practical application in knowledge acquisition tools. In particular, the diagrams form the basis of graphic knowledge editors providing a visual, formal language for entering and editing knowledge structures. For example, Figure 10 shows the minimal solution to Cendrowska's (1987) contact lens problem graphed in the graphic editor associated with KRS, our knowledge representation server (Gaines, 1990).

The three bipolar constructs and three tripolar constructs involved are graphed at the top of Figure 10. The five default rules are graphed in the center, and two individual cases are graphed at the bottom. The convention in the grapher is that concepts are represented by ovals, indviduals by rectangles, roles (which are not used in this example) by unboxed text, and special expressions (such as cardinality constraints) by rounded-corner rectangles. The lines and arrows have exactly the interpretation given in this paper, a plain line denotes a disjoint relation between primitive concepts, an arrow from one concept to another denotes that the second concept subsumes the first, the double arrow for a rule is broken with an expression "Rule" for perspicuity. Primitive concepts are indicated by small interior lines on the left and right sides which line up with the line indicating a disjoint relation in a construct.

The graphic editor itself operates in a very similar way to MacDraw with additional features taking advantage of the defined syntax and semantics of the knowledge structures to simplify data entry, for example text is automatically wrapped to as rectangular a shape as possible, boxes are automatically fitted to text, lines reconnect when items are dragged, and arcs between boxes cannot be entered if they violate the syntax. Knowledge structures in KSS0 grids and rules (Gaines & Shaw, 1987), or in KRS knowledge bases (Gaines, 1990), can be output to the graphic editor, edited and returned to KSS0 or KRS.

Since the knowledge structures in the graphic editor conform exactly to the formal schema defined in this paper it is possible to compile them directly to operational knowledge structures that will run as knowledge-based systems in a suitable shell. Thus the graphic editor provides a *visual programming language* for knowledge-based systems, one with a clear inheritance from semantic nets but now having a simple and well-defined formal semantics.

Figure 11 shows the output when the graph in Figure 11 is compiled for KRS (which provides a KL-ONE/CLASSIC-like language interface. Primitives are defined as a list of the previously defined disjoint concepts followed by the name of the new primitive to be defined. Non-primitive concepts and individuals are defined by the name followed by the defining concepts. Rules are defined by the name of the concept defining the recognition of the left-hand side followed by the concepts to be asserted on the right-hand side. The knowledge base output in Figure 11 runs directly in KRS and infers the lens recommendation for the individuals specified.

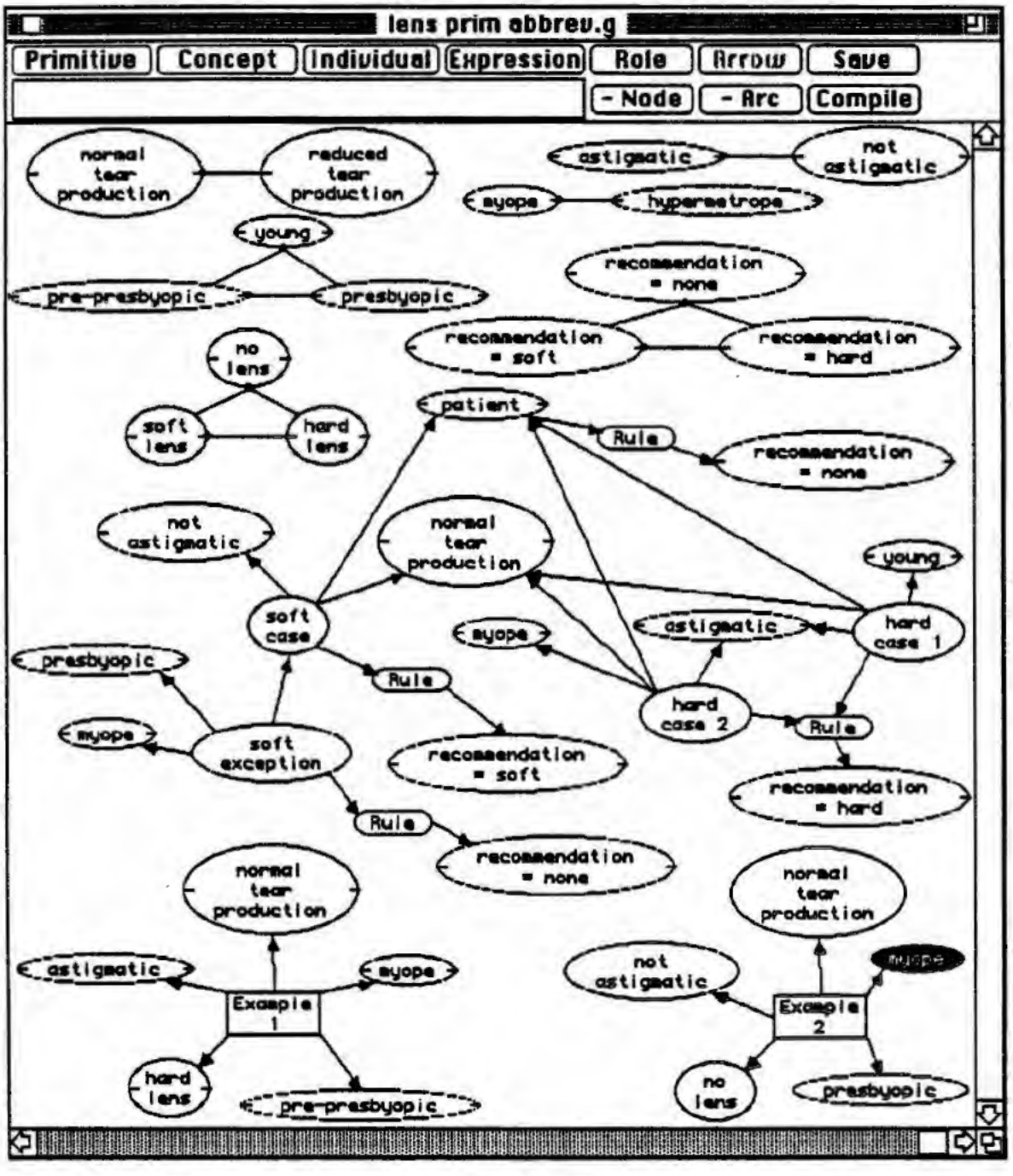


Fig.10 Graphic knowledge editor

```
Primitive(patient)
Primitive(normal tear production)
Primitive(normal tear production, reduced tear production)
Primitive(astigmatic)
Primitive(astigmatic, not astigmatic)
Primitive(myope)
Primitive(myope, hypermetrope)
Primitive(young)
Primitive(young, pre-presbyopic)
Primitive(young, pre-presbyopic, presbyopic)
Primitive(no lens)
Primitive(no lens, soft lens, hard lens)
Primitive(no lens, soft lens)
Primitive(recommendation = none)
Primitive(recommendation = none, recommendation = soft)
Primitive(recommendation = none, recommendation = soft, recommendation = hard)
Concept(hard case 1, patient, normal tear production, astigmatic, young)
Concept(hard case 2, patient, normal tear production, astigmatic, myope)
Concept(soft case, patient, normal tear production, not astigmatic)
Concept(soft exception, presbyopic, soft case, myope)
Rule(patient, recommendation = none)
Rule(hard case 1, recommendation = hard)
Rule(hard case 2, recommendation = hard)
Rule(soft case, recommendation = soft)
Rule(soft exception, recommendation = none)
Individual(Example 1, normal tear production, astigmatic, myope, pre-presbyopic, hard lens)
Individual(Example 2, normal tear production, not astigmatic, myope, presbyopic, no lens)
```

Fig.11 Visual knowledge base in the graphic editor compiled for use in the knowledge representation server

Note in Figure 11 the way in which the top-down specification of formal knowledge structures leads to a definitional assymetry in the disjoint relation; **reduced tear production** is defined as disjoint to the already-defined **normal tear production**. No logical assymetry is introduced and definition (2) still applies, but a natural psychological assymetry is replicated: that we define opposition in terms of an existing concept. Although the disjoint relation is logically symmetric, it is, at least initially, psychologically assymetric. In terms of Hegel's (1812) *Science of Logic*, the initial concept is the *thesis*, its disjoint opposition is the *antithesis*, and the resultant role representing the dimension of variation is the *synthesis*.

The compiler checks for circular definitions and reports them as errors which must be rectified before the knowledge structure is used. It automatically sorts the definitions according to dependencies so that the concepts are specified in top-down form for a knowledge-base. This is, in itself, a significant support for knowledge elicitation since the visual knowledge structures may be developed in whatever way is natural, top-down, bottom-up or middle-out, and the top-down definitional requirement that is a major problem in knowledge base development (Morik, 1987) is no longer a burden.

In addition, the compiler operates in the same way over collections of graphs allowing pre-existent concept definitions to be simply included. Thus a library of general and specific domain and task ontologies may be supported.

INTEGRATION OF KNOWLEDGE ACQUISITION METHODOLOGIES

The foundations for personal construct psychology and knowledge representation developed in this paper also provide a framework for the analysis of knowledge acquisition techniques and the design of integrated tools. Figure 12 shows the primary techniques currently in use:

- **Domain ontology** techniques cited in the introduction elicit knowledge directly as an ordinal structure of concepts defined in terms of collections of roles and constraints. This is an efficient approach to the development of knowledge structures if the knowledge is already overtly available as it is in many technical domains. As shown above, the graphic representation of the subsumption, disjunction and rule relations between concepts, and the internal roles and constraints defining them, is a natural way to represent a knowledge structure, whether directly or indirectly elicited. Some form of graphic editing tool for presenting and editing domain ontologies is an essential component for an integrated knowledge acquisition tool.
- **Repertory grid** techniques cited in the introduction elicit knowledge indirectly by prompting the experts for critical cases and relevant constructs in a coherent sub-domain. The constructs define a domain concept that acts as a templet for the cases which are expressed as individuals recognized by the domain concept. Figure 13 shows how the grid relates to the conceptual structure of a domain, a grid is that structure which results from combining a concept with the individuals that it distinguishes. Thus, there is a grid corresponding to each domain concept or sub-domain. This corresponds to the application of grid tools in rapid prototyping to characterize a number of significant sub-domains in detail. The significance of grid techniques is that they provide a method of indirectly developing domain concepts through examples *when direct methods based on overt knowledge are not available.*
- **Empirical Induction** techniques such as C4.5 (Quinlan, 1987), Prism (Cendrowska, 1987) and Induct (Gaines, 1989) take a database of cases and derive constraints upon it that correspond to rules in knowledge representation schema. Each side of a rule defines a concept in terms of roles and constraints and these form a natural sub-domain concept structure as shown at the bottom of Figure 8. There are no fundamental differences between the application of induction techniques to a small database of correctly described critical cases expressed in terms of relevant attributes in a grid, and to a large database of lower quality data containing errors and irrelevant attributes. There is a strong trade-off between the quality of data and the amount of data required to induce the same quality of knowledge with knowledge acquisition tools emphasizing the elicitation of high-quality data, and empirical induction tools emphasizing the analysis of large datasets (Gaines, 1989).

The three techniques described come together naturally to give an integrated system as shown in Figure 14.

Stage ① consists of developing a basic domain ontology focusing on determining the major sub-domains rather than the detailed elicitation of the complete domain. Unstructured interviews, group synectics, text and protocol analysis techniques are appropriate to this stage in developing a broad and complete picture of the major features of a domain (Shaw & Woodward, 1989).

Stage ② consists of eliciting grids for each major sub-domain in order to characterize them in detail in terms of relevant constructs and critical cases. This enables the templet concept for the sub-domain to be developed together with cases for further analysis.

Stage ③ consists of using inductive analysis of the cases to derive constraints providing a conceptual structure and rules for the sub-domain. This can be evaluated by a sub-domain expert as an overt knowledge structure and in terms of its performance on additional cases, and edited and further developed as necessary.

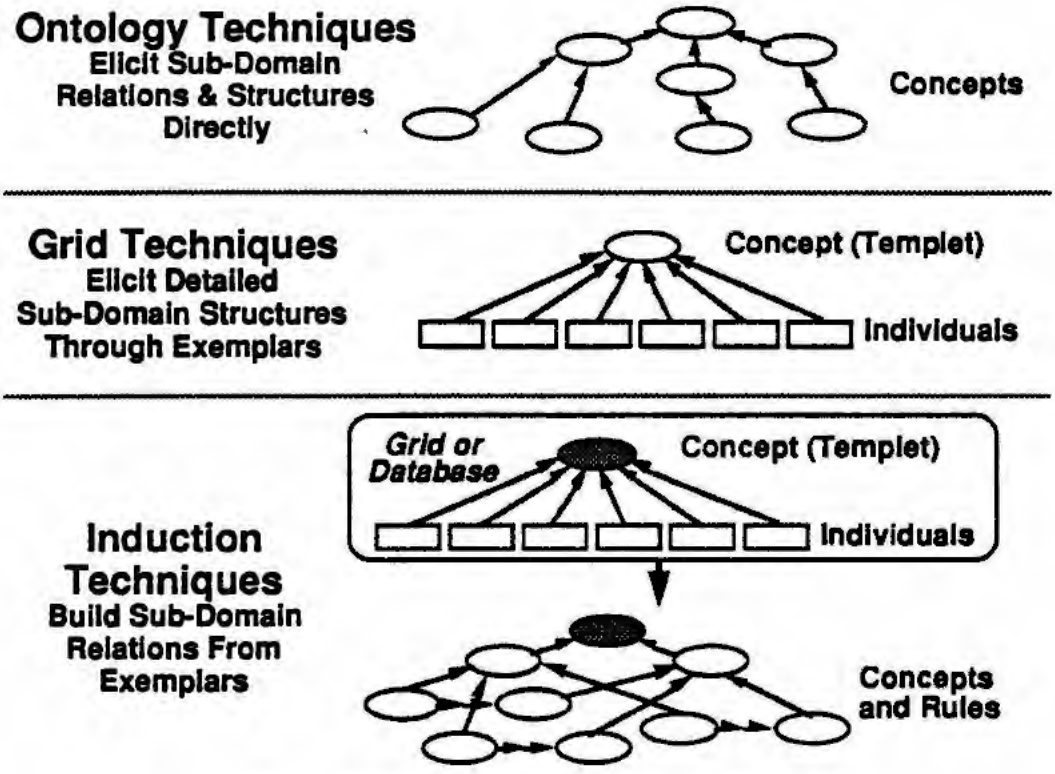


Fig.12 Knowledge acquisition techniques

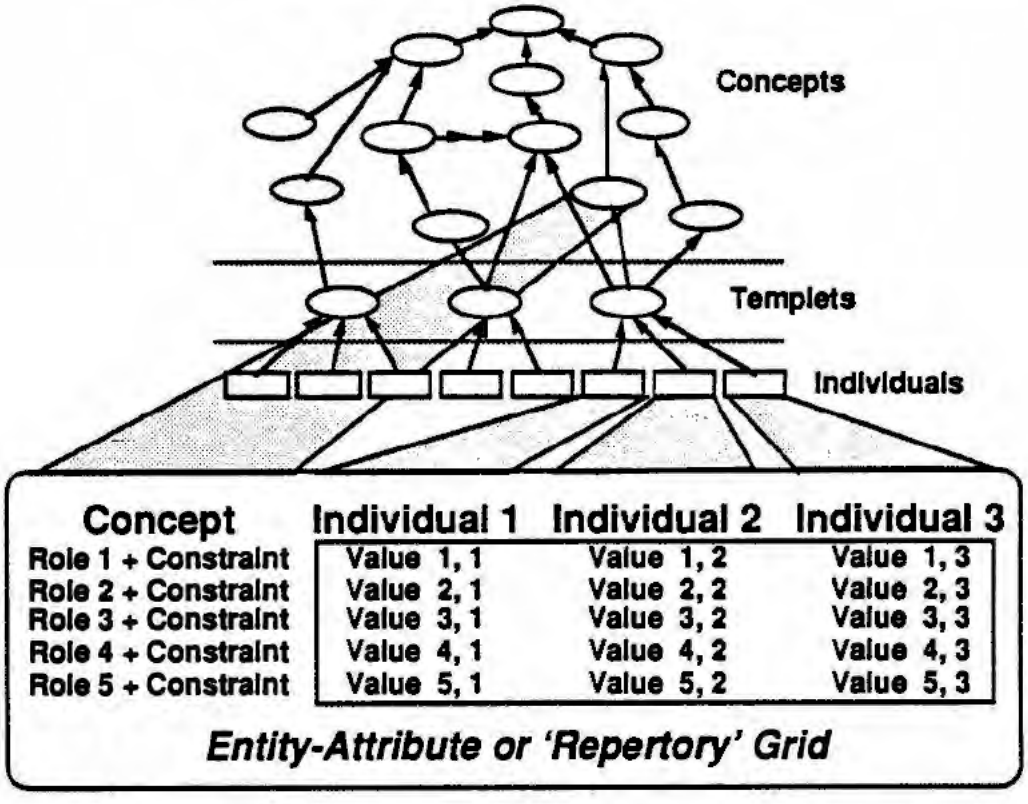


Fig.13 The repertory grid: individuals in the coordinate system of a concept

Stage ④ consists of combining the sub-domain knowledge structures to form an overall knowledge-based system for the domain. This will usually consist of linking the sub-domains through individuals in one sub-domain that are values of roles in others.

Stage ⑤ consists of testing the overall knowledge base and iterating back through any of the earlier stages in order to develop and refine it further. Note that the integrative, homogeneous framework for the overall process makes this a reasonable exercise.

This knowledge acquisition schema is an extrapolation and integration of existing, well-developed techniques for knowledge acquisition. It is particularly significant in providing means for the integration of acquisition and performance, supporting ongoing development of knowledge-based systems as part of their application. Each case for which the system is used provides new examples within the sub-domains that may be incorporated within the grids and used for further development. There is no intrinsic distinction between acquisition and performance.

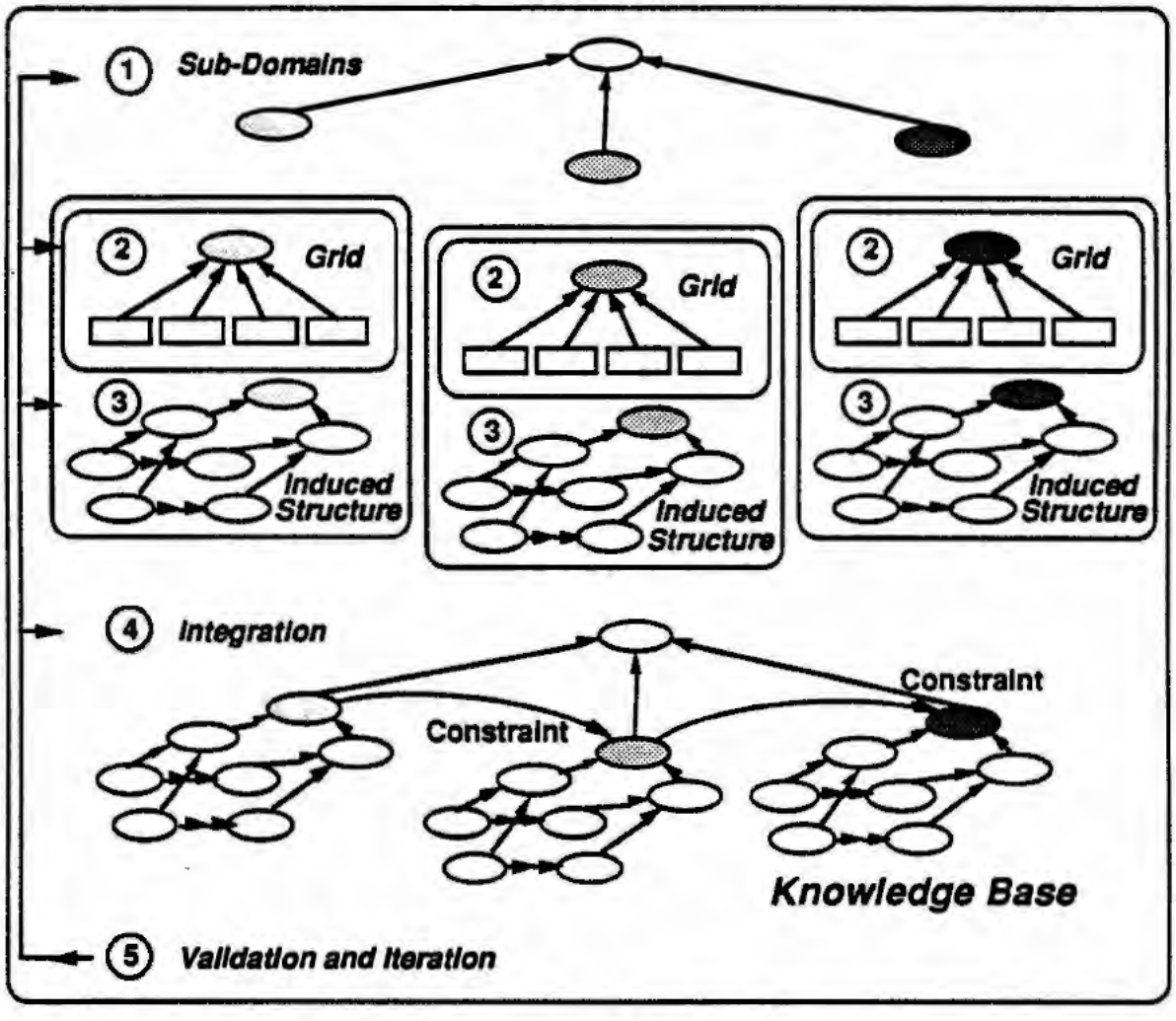


Fig.14 Integration of techniques

Other tools have a natural place in the systems of Figures 12 and 14. Conceptual clustering techniques such as FOCUS and PrinCom (Shaw & Gaines, 1987) are an alternative way of developing concepts by grouping related constructs together. They are attractive in allowing the expert to visualize conceptual relationships and see how they derive from the cases.

Techniques for comparing the knowledge structures of different experts, such as Socio (Shaw & Gaines, 1989), are important in highlighting conflicts in the use of distinctions and terminology and in enriching concepts by providing additional distinctions and alternative ways of expressing them. This analysis is related to Kelly's *commonality corollary*, that similar construction systems generate similar psychological processes.

The individual and attribute hierarchies introduced in AQUINAS (Boose & Bradshaw, 1987) are intrinsically available within the knowledge representation framework developed in this paper. They correspond to the extension of the grid in Figure 13 to encompass multiple related concepts.

Abstract individuals in the grid, such as an *ideal entity*, cannot be adequately represented as concrete individuals. They are basically a system of constraints not values and correspond to a concept rather than an individual. However, it is possible to provide a user interface for the definition of such abstract individuals in terms of constraints which is very similar to that for the entry of data about existent individuals in terms of values.

Such abstract individuals also provide the mechanism for decision-making based on pattern-matching as used in AQUINAS' inbuilt performance engine. Cases are generalized into concepts which can be used to recognize similar cases in future.

CONCLUSIONS

The main features of personal construct psychology have been presented. A theoretical framework has been developed and shown to provide logical foundations for personal construct psychology and computational knowledge representation schema. The framework is generated from the single primitive of "making a distinction." Figure 15 recapitulates the main definitions derived from this primitive.

Distinction	Primitive notion
Subsumes	Asserting one distinction commits one to asserting another
Disjoint	Asserting one distinction commits one to not asserting another
Individual	Initial distinction that cannot be asserted of others
Concept	Mental entity imputed to agent to enable it to make a distinction
Construct	Triple of two disjoint concepts subsumed by a third *(dual* to concept)
Role	Internalization of a construct within a concept
Value	Concept in role applicable to an individual
Non-Primitive	Concept defined in terms of roles and applicable concepts
Recognize	Individual conforming to definition of a non-primitive concept
Rule	Commitment to assert an additional concept of a recognized individual

Fig.15 Main definitions in paper

The primitives derived in Figure 15 have been used to provide cognitive and logical foundations for existing knowledge acquisition tools and techniques, and for the design of integrated knowledge acquisition systems. A companion paper (Gaines, 1990) describes a practical implementation of such a system.

It is interesting to consider in conclusion what this paper leaves out, the *sources* of distinctions. Two paradigms not based on overt knowledge representation are both relevant. Situated action (Suchman, 1987; Clancey, 1989) explicates behavior in terms of a total systemic relation between the environment and the individual, including the inheritance of distinctions embedded in the built environment. *Neural networks* (Smolensky, 1986) provide a computational mechanism for creating new major distinctions at the symbolic level from what is virtually a continuum of distinctions at the sub-symbolic level. The framework presented in this paper has natural extensions to include these other paradigms and, potentially, give a complete account of human knowledge processes.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. We are grateful to many colleagues for discussions over the years that have influenced the research described in this paper, in particular John Boose, Jeff Bradshaw, Bill Clancey, Gordon Pask, Alain Rappaport, Doug Skuce, Brian Woodward and other colleagues at the Knowledge Acquisition Workshops world-wide.

REFERENCES

Anglin, J.M. (1977) **Word, Object, and Conceptual Development.** Norton: New York.

Bannister, D. & Mair, J.M.M. (1968) **The Evaluation of Personal Constructs.** London: Academic Press.

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. (1986). **Expertise Transfer for Expert Systems.** Amsterdam: Elsevier.

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.

Borgida, A., Brachman, R.J., McGuiness, D.L. & Resnick, L.A. (1989). CLASSIC: a structural data model for objects. Clifford, J., Lindsay, B. & Maier, D., Eds. **Proceedings of 1989 ACM SIGMOD International Conference on the Management of Data.** pp.58-67. New York: ACM Press.

Brachman, R.J. (1983). What IS-A is and isn't. **IEEE Computer, 16**(10) 30-36 (October).

Brachman, R.J. & Schmolze, J. (1985). An overview of the KL-ONE knowledge representation system. **Cognitive Science, 9**(2) 171-216 (April-June).

Bradshaw, J.M., Boose, J.H., Covington, S.P. & Russo, P.J., (1988) How to do with grids what people say you can't. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.3-1-3-15. Banff (November).

Breuker, J. & Wielinga, B. (1987) Use of models in the interpretation of verbal data. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum.

Brown, G.S. (1969) **Laws of Form.** London: George Allen & Unwin.

Bruck, R.H. (1971) **A Survey of Binary Systems.** Berlin: Springer-Verlag.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Clancey, W.J. (1990) The frame of reference problem in the design of intelligent machines. van Lehn, K.& Newell A. (Eds.) **Architectures for Intelligence: The Twenty-Second Carnegie Symposium on Cognition.** Hillsdale: Lawrence Erlbaum.

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.

Eshelman, L., Ehret, D., McDermott, J. & Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies, 26**(1), 41-54

Ford, K.M., Stahl, H., Adams-Webber, J., Novak, J. & Jones, J.C. (1990) An integrated constructivist knowledge acquisition tool. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.7-1-7-20. Banff (November).

Frege, G. (1892). On sense and reference. Geach, P. & Black, M., Eds. **Translations from the Philosophical Writings of Gottlieb Frege.** pp.56-78. Oxford: Basil Blackwell, 1970.

Gaines, B.R. (1976). Foundations of fuzzy reasoning. **International Journal Man-Machine Studies, 8**, 623-668.

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. & 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. (1981). A programme for the development of a systems methodology of knowledge and action. Reckmeyer, W.J., Ed. **General Systems Research and Design: Precursors and Futures.** pp. 255-264. Society for General Systems Research.

Gaines, B.R. & Shaw, M.L.G. (1984). Hierarchies of distinctions as generators of system theories. Smith, A.W., Ed. **Proceedings of the Society for General Systems Research International Conference.** pp. 559-566. California: Intersystems.

Gaines, B.R. & Shaw, M.L.G. (1986). Induction of inference rules for expert systems. **Fuzzy Sets and Systems, 18**(3), 315-328 (April).

Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. 47-66.

Gracia, J.J.E. (1983) Individuals as instances. **Reviews of Metaphysics, 37**(1), 37-59 (September).

Garg-Janardan, C. & Salvendy, G. (1987). A conceptual framework for knowledge elicitation. **International Journal of Man-Machine Studies, 26**(4), 521-531.

Grossmann, R. (1983) **The Categorical Structure of the World.** Bloomington: Indiana University Press.

Hegel, W.F. (1812) **Science of Logic.** Johnston, W.H. & Struthers, L.G., Trans. (1929) London: George Allen & Unwin.

Hinkle, D.N. (1965). The change of personal constructs from the viewpoint of a theory of implications. PhD Thesis, Ohio State University.

Jones, W.P. (1989) Bringing corporate knowledge into focus with CAMEO. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.17-1-17-23. Banff (October).

Kelly, G.A. (1955). **The Psychology of Personal Constructs.** Norton: New York.
Klinker, G., Bentolila, J., Genetet, S., Grimes, M. & McDermott, J. (1987) KNACK—report-driven knowledge acquisition. **International Journal of Man-Machine Studies, 26**(1), 65-79.
Lloyd, G.E.R. (1966) **Polarity and Analogy.** Cambridge: Cambridge University Press.
Mac Lane, S. (1971) **Categories for the Working Mathematician.** New York: Springer-Verlag
Mac Lane, S. (1986) **Mathematics Form and Function.** New York: Springer-Verlag
Marcus, R.B. (1962). Interpreting quantification. **Inquiry, 5**, 252-259.
Marcus, S. (1987). Taking backtracking with a grain of SALT. **International Journal of Man-Machine Studies, 26**(4), 383-398.
Maybury-Lewis, D. & Almagor, U. (1989). (Eds) **The Attraction of Opposites.** Ann Arbor: University of Michigan Press.
Meinong, A. (1910). **On Assumptions.** Berkeley: University of California Press, 1983.
Monarch, I. & Nirenburg, S. (1987) The role of ontology in concept acquisition for knowledge-based systems. **Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'87).** Reading University, UK (September).
Morik, K. (1987) Sloppy modeling. Morik, K. Ed. **Knowledge Representation and Organization in Machine Learning.** pp.107-134. Berlin: Springer-Verlag.
Motta, E., Eisenstadt, M., Pitman, K. & West, M. (1988) Support for knowledge acquisition in the knowledge engineer's assistant (KEATS). **Expert Systems 5**(1), 6-28.
Pask, G. (1975). **Conversation, Cognition and Learning,** Elsevier, Amsterdam.
Quinlan, J.R. (1987) Simplifying decision trees. **International Journal of Man-Machine Studies 27** (3), 221-234 (September).
Russell.B. (1905). On denoting. **Mind, 14**, 479-493 (October).
Sanford, D.H. (1975). Borderline logic. **American Philosophical Quarterly, 12**, 29-39.
Shaw, M.L.G. (1979). Conversational heuristics for enhancing personal understanding of the world. Gaines, B.R., Ed. **General Systems Research: A Science, A Methodology, A Technology.** pp. 270-277. Kentucky: Society for General Systems Research.
Shaw, M.L.G. (1980). **On Becoming A Personal Scientist: Interactive Computer Elicitation of Personal Models Of The World.** London: Academic Press.
Shaw, M.L.G. & Gaines, B.R. (1979). Externalizing the personal world: computer aids to epistemology. **Improving The Human Condition: Quality and Stability in Social Systems.** pp. 136-145. Kentucky: Society for General Systems Research.
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. (1989) Mental models in the knowledge acquisition process. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.29-1-29-24. Banff (October).
Skuce, D., Shenkang, W. & Beauville, Y. (1989) A generic Knowledge Acquisition Environment for conceptual and ontological analysis. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.31-1-31-20. Banff (October).

Smolensky, P. (1986). Information processing indynamical systems: foundations of harmony theory. Rumerhart & D.E., McClelland, J.L., Eds. **Parallel Distributed Processing 1.** pp.194-263. Cambridge, Massachusetts: MIT Press.

Suchman, L.A. (1987). **Plans and Situated Action.** Cambridge: University Press.

Woods, J. (1974). **The Logic of Fiction.** The Hague: Mouton.

Woods, W.A. (1975). What's in a link: foundations for semantic networks. Bobrow, D.G. & Collins, A.M. **Representation and Understanding: Studies in Cognitive Science.** pp.35-82. New York: Academic Press.

Woodward, B. (1990) Knowledge engineering at the front-end: defining the domain. **Knowledge Acquisition** 2(1), 73-94 (March).

Zadeh, L.A. (1964) The concept of state in system theory. Mesarovic, M.D. (Ed.) **Views on General Systems Theory: Proceedings of the Second Systems Symposium at Case Institute of Technology.** New York: John Wiley.

Zadeh, L.A. (1965). Fuzzy sets. **Information and Control, 8,** 338-353.

Zalta, E.N. (1983). **Abstract Objects.** Dordrecht: Reidel.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

ENVISIONING GROUP KNOWLEDGE ACQUISITION: FOCUS GROUPS IN THE CONTEXT OF A MIXED KNOWLEDGE ACQUISITION METHODOLOGY

Martha Grabowski

Le Moyne College
Syracuse, New York 13214
bitnet: grabowsk@lemoyne

Department of Decision Sciences and Engineering Systems
Center for Industrial Innovation 8211
Rensselaer Polytechnic Institute
Troy, NY 12180

Anne Massey
William A. Wallace

Department of Decision Sciences and Engineering Systems
Center for Industrial Innovation 5015
Rensselaer Polytechnic Institute
Troy, NY 12180
internet: awallace@dses1.eng.rpi.edu

ABSTRACT

Architectures and methodologies provide knowledge acquisition researchers and practitioners broad working frameworks within which to explore a variety of questions. However, in order to rigorously pursue the development of robust reasoning systems with reasonable expectations of the development process, empirical assessments of the knowledge acquisition process and of heuristic inputs to the process are essential. This paper describes the addition of group knowledge acquisition techniques to a repertoire of techniques previously referred to as a 'mixed knowledge acquisition methodology.' In this paper we discuss an experiment whose results support the use of a group knowledge acquisition technique in combination with other knowledge acquisition techniques.

1.0 INTRODUCTION

Industrial and academic developers of expert systems are searching for methodologies which provide appropriate guidance for the development of expert systems. One piece of this puzzle is the search for a knowledge acquisition methodology which spans both research and commercial environments, amenable to a variety of implementations. Our previous research provided empirical support for the anecdotal advice that use of a mix of knowledge acquisition methods was important [Grabowski90]. However, our earlier work only evaluated single-expert knowledge acquisition techniques. This paper describes additional work done within the context of empirically evaluating the "mixed knowledge acquisition methodology" to include multiple expert paradigms.

2.0 KNOWLEDGE ACQUISITION--A PRELIMINARY MODEL

A high-level description of the front-end steps in building expert systems is given in Figure 1 [Grabowski90]. These steps are cast within a familiar Identification-Conceptualization-Formalization framework [Hayes-Roth, et.al.83], and four phases: a preliminary phase, followed by three knowledge acquisition (KA) phases. The preliminary tasks establish the task, requirements, organization, software and hardware framework of the project. Phase 1 KA tasks (Identification) follow the preliminary KA phase, and similarly establish a structure--between the knowledge engineer and the domain expert--by relying on traditional information requirements determination methods (interviews, questionnaires, and observation). Phase 2 KA (Conceptualization) offers a wide variety of KA techniques, which are used to "fill in the gaps" in the system's knowledge base and reasoning. Phase 3 KA (Formalization) begins the transition from knowledge acquisition to knowledge representation. In Phase 3, knowledge engineers can use iterative prototyping, traditional systems development methodologies, or a hybrid development approach incorporating elements of both models.

2.1 Preliminary Knowledge Acquisition

In many ways, the preliminary KA phase resembles the preliminary steps of traditional systems development. Like traditional systems development, knowledge acquisition begins with a problem identification phase, which focuses the problem to be solved. Once the problem is identified and scoped, users and customers are identified, and good working relationships are established. Domain experts are identified, and determined to be available, interested, and able to be committed. Background reading is done prior to beginning formal interviews with the expert(s).

Armed with a focused problem, a good working relationship with the customer/user, identified (and available) domain experts, and an understanding of the domain and problem, the knowledge engineer embarks upon a requirements analysis to identify the functions which need to be satisfied with the system, which dictates the tasks the system must support. A task analysis may form the basis for the requirements analysis. Man-machine tradeoff studies are often performed during the requirements analysis in order to assess which tasks and requirements will be supported by people, and which by machines.

An organizational analysis is the next step in the process. The organizational analysis determines how the man and machine tasks of the future will be supported in the environment. A software analysis follows the organizational analysis. The software analysis focuses on the information processing requirements of the tasks allocated to machines: which of the machine tasks can be supported with conventional software, and which are more amenable to a decision support systems or expert systems implementation? The results of the task, requirements, man-machine tradeoff and software analyses are graphically represented in a software architecture matrix, which identifies the tasks to be supported, and the mapping between the machine tasks and the appropriate software implementation--information system, decision support system, or expert system.

A hardware analysis follows the software analysis. During the hardware analysis, system development and deployment hardware is identified, as are any constraints imposed. The result is the system's hardware architecture, now driven by the system requirements, the tasks to be supported, a recognition of the organizational and environmental impacts on the system, and the underlying software architecture.

At this point, the knowledge engineer may make one final analysis before interacting with the domain expert(s): a preliminary separation of the generic domain information from the site-specific knowledge and from the meta- (control) knowledge. The results of preliminary knowledge acquisition tasks, then, are a focused problem, identified and committed knowledge

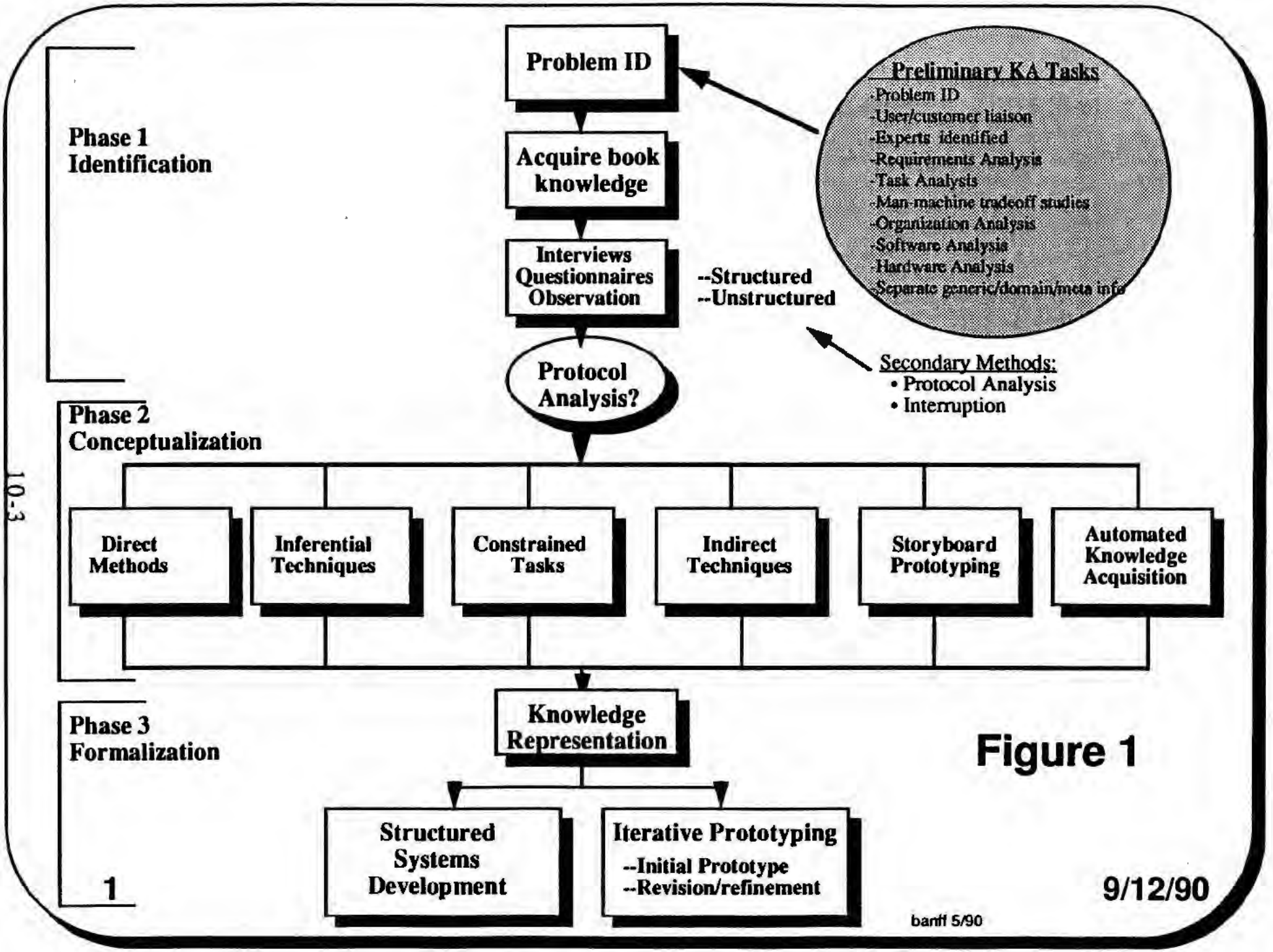

Phase 1
Identification
Problem ID
Acquire book knowledge
Interviews
Questionnaires
Observation
--Structured
--Unstructured
Protocol Analysis?
Preliminary KA Tasks
-Problem ID
-User/customer liaison
-Experts identified
-Requirements Analysis
-Task Analysis
-Man-machine tradeoff studies
-Organization Analysis
-Software Analysis
-Hardware Analysis
-Separate generic/domain/meta info
Secondary Methods:
• Protocol Analysis
• Interruption
Phase 2
Conceptualization
Direct Methods
Inferential Techniques
Constrained Tasks
Indirect Techniques
Storyboard Prototyping
Automated Knowledge Acquisition
Phase 3
Formalization
Knowledge Representation
Structured Systems Development
Iterative Prototyping
--Initial Prototype
--Revision/refinement
1
9/12/90
banff 5/90

Figure 1

engineers and experts, and the software, hardware, organizational, and domain context needed for the remainder of the system building process.

2.2 Phase 1 Knowledge Acquisition

Once the preliminary knowledge acquisition tasks are completed, Phase 1 knowledge acquisition begins. Phase 1 KA looks very much like the information requirements determination (IRD) stages of structured systems analysis and design [Davis83, Senn89]. Interviews, questionnaires, and observation of experts--all traditional means of IRD for conventional software development--are the techniques used during Phase 1, since Phase 1 KA is the period during which the knowledge engineer and the expert forge their relationship. The goal is to establish a common domain framework and a strong relationship between the knowledge engineer and the expert, and conventional IRD techniques provide good support for these "structure-building" tasks. At the tail end of Phase 1, the knowledge engineer decides whether or not to use protocol analysis--the process of translating a subject(s)' "thinking aloud" verbalizations into a map of the subject's decision-making thought sequences [Newell and Simon72; Ericsson and Simon84].

Preliminary and Phase 1 KA, then, (1) begin with a problem narrowing, (2) proceed with task, requirements, organizational, software and hardware analyses, and (3) conclude with a problem-widening process (interviews, questionnaires, observation, and traditional IRD techniques). The information gathered during the preliminary and Phase 1 phases is modeled and refined during the next stage, Phase 2 knowledge acquisition.

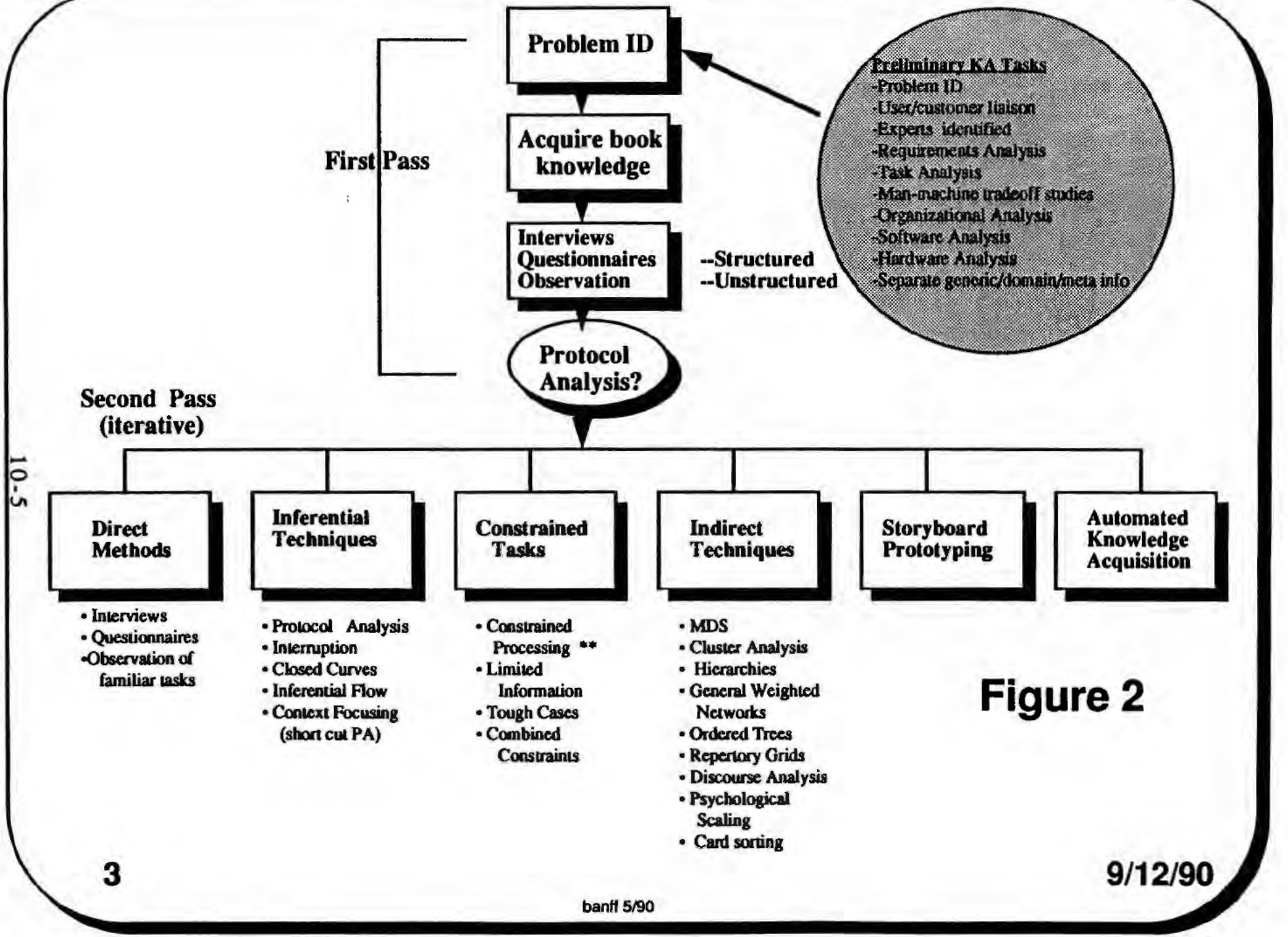
2.3 Phase 2 Knowledge Acquisition

Phase 2 KA offers a wide variety of non-traditional techniques to acquire information not already elicited during Phase 1. The knowledge engineer and the expert will rely on the relationship and context developed during Phase 1, as both will be strained with the variety, complexity, and ambiguity of Phase 2 KA techniques. The danger of a poor Phase 1 KA phase becomes obvious during Phase 2--if the knowledge engineer and the expert have not established a strong foundation, the project may come to a grinding halt under the Phase 2 stresses--new techniques, pressures for new or missing information, and unfamiliarity with "soft," qualitative methods.

During Phase 2 KA, a variety of graphical, qualitative, and analytical techniques are used: direct methods, inferential techniques, constrained processing tasks, indirect techniques, prototyping, and automated knowledge acquisition techniques (Figure 2). These methods are surveyed in detail in [Reitman Olson and Reuter87] and [Grabowski90]. The goal of Phase 2 KA is to use these multitude of techniques to fill in the gaps in the knowledge base or reasoning processes and to elicit "information delta's": heuristics, knowledge, beliefs, and information not previously elicited. Phase 2 KA stretches and strains the knowledge engineer and the expert, by relying on "softer" methods to "round out" the knowledge base by eliciting deeper domain information [Leifer, Durgee and Wallace89].

2.4 Phase 3--System Development/Iterative Prototyping

During Phase 3, expert systems prototypes are constructed. Phase 3 offers the knowledge engineer a choice--iterative prototyping for the more uncertain expert system components, and traditional systems analysis and design methods for the more conventional software portions [Weitzel and Kershberg89]. The characteristics of the domain information, requirements, tasks, and environment (with respect to ambiguity, completeness, and uncertainty) determine which method is appropriate. This delineation is not dissimilar to uncertainty analyses in conventional IRD [Davis and Olson85].


Problem ID
Acquire book knowledge
Interviews
Questionnaires
Observation
--Structured
--Unstructured
Protocol Analysis?
First Pass
Preliminary KA Tasks
-Problem ID
-User/customer liaison
-Experts identified
-Requirements Analysis
-Task Analysis
-Man-machine tradeoff studies
-Organizational Analysis
-Software Analysis
-Hardware Analysis
-Separate generic/domain/meta info
Second Pass (iterative)
Direct Methods
• Interviews
• Questionnaires
•Observation of familiar tasks
Inferential Techniques
• Protocol Analysis
• Interruption
• Closed Curves
• Inferential Flow
• Context Focusing (short cut PA)
Constrained Tasks
• Constrained Processing **
• Limited Information
• Tough Cases
• Combined Constraints
Indirect Techniques
• MDS
• Cluster Analysis
• Hierarchies
• General Weighted Networks
• Ordered Trees
• Repertory Grids
• Discourse Analysis
• Psychological Scaling
• Card sorting
Storyboard Prototyping
Automated Knowledge Acquisition

10-5

Figure 2

Phase 3 KA begins the transition process from knowledge acquisition to knowledge representation. As KA winds down, and Phase 3 software development begins, the knowledge engineer becomes less concerned with eliciting large volumes of information and more concerned with (1) identifying and modifying "holes" in the knowledge base and system reasoning, and (2) adequately representing the elicited information in a machine-implementable coding scheme.

3.0 THE NEED FOR THE ADDITION OF A MULTIPLE EXPERT PERSPECTIVE

A recent addition to our Phase 2 KA repertoire is the use of group knowledge acquisition techniques. This paper describes our first entree into explorations of group KA techniques--an empirical assessment of focus groups as a means of facilitating group knowledge acquisition.

Organizational studies show that groups perform better than individuals in complex tasks because they present a broader base of information and expertise. Groups often produce more and better solutions than individuals because more information becomes available, and members can reject incorrect suggestions and correct errors–the concept of collective rationality [Fisher74]. Collins and Guetzkow [64] discuss the "assembly-effect bonus," which holds that a group can produce better results than would be expected, given the individual characteristics of the individual members.

Group knowledge acquisition has a number of advantages over individual knowledge acquisition sessions [Massey and Wallace90]:

- they can provide broader ranges of skills and knowledge,
- they can provide more effective divisions of labor, and
- they can legitimize a result.

However, sometimes the "costs: of group KA sessions may outweigh the benefits [Massey and Wallace90]:

- groups may not fully or efficiently utilize the knowledge they contain,
- influential members may control the group,
- pressures to agree or conform can adversely affect free expression and divergent thinking, and
- the diversity of knowledge within the group can make consensus difficult to achieve.

3.1 Previous Research

There has been little empirical research conducted to develop robust KA methodologies, or to evaluate different knowledge acquisition techniques. Some frameworks and surveys in knowledge acquisition are given in Benbasat and Dhaliwal [88] and Boose [88]; and Grover [83], Hoffman [87], Prerau [87], and Cooke and McDonald [86] evaluate different set of KA techniques. Early work in evaluating knowledge acquisition techniques is summarized in [Dhaliwal and Benbasat89]. There have been a number of studies evaluating different automated knowledge acquisition tools [Burton, et.al.,87; Shaw and Woodward88; Kitto88; Linster88].

We have been concerned that earlier paradigms of the mixed knowledge acquisition methodology described were (1) limited (narrow) and (2) devoid of any mention of multiple knowledge engineer or multiple expert interactions. Yet, our experience in developing reasonably sized, robust expert systems for aerospace [ALBM89] and commercial applications [Grabowski87] reminds us of the importance of including group techniques. Our users and customers increasingly demand that more than one subject matter expert (SME) be involved in our KA sessions. Thus, we embarked on the work described in this paper, which sought to empirically evaluate group KA perspectives

and approaches within the context of our mixed KA methodology. As a first step in this process, we investigated focus groups as a group KA technique.

3.2 Focus Groups

Focus groups are informal discussions among a small group of individuals, ideally five to nine people. These discussions, typically of two hours' duration, are focused by a skilled moderator on a particular topic. In marketing research, these topics are typically a product or service. The degree of structure imposed on the discussion, as well as the composition of the group, is a function of the objective of the session. The moderator is key to the session. He or she stimulates discussion and maintains focus on the topic without biasing the responses of the participants. The moderator is not an expert in the discussion; the participants are.

Moderated small group discussions about a topic are frequently used as a method of qualitative research in marketing [Advertising Research Foundation85]. These "focus groups" are not new; they were used by researchers in the 1940's to study wartime propaganda [Merton, Fiske and Kendall56]. The approach is based on a desire to understand the participants' behavior by gaining insights into motivations that are difficult to express [Goldman62]. Morgan [88] reviews focus groups as qualitative research methods.

A focus group is, in reality, a group of individuals who have come together to address a particular problem or task. The focus group is not expected to reconvene in the future, and it is assumed that they have not acted as a group in the past. Focus groups are, therefore, one instance of non-repeating group behavior [Zmud88]. Non-repeating group behavior consists of two distinct processes: relation-building and decision-resolution. The extent to which these processes are evident depends on the purpose of the group. In the case of a focus group, one is not as interested in reaching a decision, as in establishing the environment that facilitates group interaction. We are concerned with stimulating divergent thinking, in order to understand the reasoning process, rather than having the group converge to a "decision" or resolution of a problem.

Focus groups may be considered as group interviews, whose purpose is to collect qualitative data. Unlike other group techniques, focus groups rely on the "explicit use of group interaction to produce data and insights that would be less accessible without the interaction in a group [Morgan88]. They are purported to provide information which is qualitatively different from that obtained by summing the individual interviews [Goldman62]. A wider range of information, insights, and ideas can be obtained from groups than individuals [Hess68].

One advantage of group interviewing over individual interviewing is that the participant interaction among themselves replaces their interaction with the interviewer. This leads to greater emphasis on the participants' point of view. For knowledge acquisition purposes, this may imply an easier and quicker method to get at "deeper" knowledge than individual interviewing, which requires the knowledge engineer to first get deeply immersed in the domain. The group members themselves may be considered knowledge engineers since they can understand and speak the domain language. This provides the capability to challenge each other and to respond to challenges from other members.

The exploratory nature of group interviewing requires much less preparation than that of either structured or unstructured interviews. The use of focus groups may be appropriate as a basis of creation of an interview guide for more groups or individual interviews [Morgan88], and thus, a first step in the preliminary knowledge acquisition phase.

The main problem facing the use of focus groups is the management of collected data. Since the moderator has less control over the group than during an interview, more "non-useful" data may be produced. Fern [82, 83] compared the number of ideas generated in focus groups and in an

equivalent number of interviews and found that groups produce roughly 70% as many ideas as individuals. However, it should be noted that if time and cost are important to an organization, the number of focus group sessions required (usually one) versus the number of individual interviews is much less (as is the analysis time for the produced transcripts). In the context of knowledge acquisition, the loss of data (identified by Fern) may not be significant if, in fact, it is shallow or general knowledge. Rather, what is desired is for the group setting (with members acting as their own "group knowledge engineers") to facilitate the exposure of deep knowledge.

This research investigates the hypothesis that face-to-face group interaction with experts in focus groups provides different "thoughts and intuitions" about a task or situation than individual interviews. Multiple experts were used in a situation assessment context. We hypothesized that focus groups may be useful in knowledge acquisition because the interaction between respondents draws more cognition and "rules" to the surface (just as group therapy draws feeling out of mental patients). The subsequent verbatim transcripts of the respondents were analyzed to discover the underlying rules and cognitive schema.

We were encouraged in an exploration of focus groups as a means of performing group KA by the focus groups' strengths: primarily, their ability to explore topics with little direction from the moderator. In addition, they are easy, inexpensive, and quick to conduct. However, on the downside, focus groups' weaknesses include the fact that the sessions are not conducted in the experts' natural settings and that there is little control over the data generated [Morgan88]. In addition, there may be a tendency for one individual in the group to dominate the discussion and sway other members' comments.

4.0 FOCUS GROUPS AS A KNOWLEDGE ACQUISITION TECHNIQUE

4.1 Experimental Setting

A banking group wished to instill a sales culture among its hundreds of branch outlets. Although sales training programs can be established in which trainees are told "what" needs to be done (i.e., expand existing business, make more calls), very few "how to do it" guidelines or rules have been identified. Without these, the implementation of a sales culture, within and among the bank's branches, is difficult.

A successful sales call requires both analysis and professional judgment. The analytic reasoning involves performing a rudimentary financial analysis, while judgment is needed to assess the likelihood of a sale and to reach a conclusion satisfactory to both the bank official and the client.

Every sales call has a common set of activities: identify the client's needs, prioritize them, relate them to the financial well-being of the client, and ascertain a "feel" for the possibility of doing business with that person or organization. Written procedures and guidelines exist for all of these activities. However, determining the "feel" for a sales call requires judgment and intuition, which was difficult for a knowledgeable sales person to articulate. Eliciting this expertise was the experimental setting used to assess face-to-face group interaction (focus groups) as a knowledge acquisition technique.

4.2 The Scenarios

The first step was to develop "book knowledge" of detailed rules and guidelines from the training material developed by bank personnel and consultants. Selling concepts that successful sales people follow exist, and form the basis of this training material. The ability to take "implied needs" of the potential buyer and develop them into "explicit needs" has been identified as the key difference between success and failure. The sales person takes on the role of a "problem

finder" for the buyer by assisting in problem formulation. "How" does this "expert" do this? What makes him/her go beyond the sales framework presented in the sales training course? Can the knowledge of these successful sales people be captured and developed into a form that can be used to better train less experienced personnel? This became the focus of the next steps in the research.

The sales training program identifies the steps to follow when making a sales call: (1) greeting or opening; (2) questioning to uncover needs and priorities; (3) reinforcing and clarifying needs; (4) presentation; and (5) closing. For each of these steps, procedural and "personality profiling" guidelines are taught in the training course. The written guidelines were used as "book knowledge"; rules were developed as abstractions from the training manual by the researchers. Based on this book knowledge and discussions with bank personnel, scenarios were developed depicting three typical sales situations. These scenarios were: (1) a company president seeking an additional credit line, (2) a company treasurer with cash management priorities, and (3) a dentist seeking a loan to set up her own practice.

Scripts were prepared for each of the scenarios and videotaped with bank personnel acting out the roles. Each of the approximately four minute taped scenarios was divided into five segments representing the sale steps identified above. One objective of using individual interviewing and conducting a focus group session is to compile, for each of the scenarios, rules or guidelines from the experts that were not found in the documentation.

4.3 The Subjects and Interviews

The ten participants in the study were all considered to be "experts" in terms of selling bank products and services. Their experience in sales was considered by bank officials to be equivalent, as were their positions in the organization. Five salespeople were interviewed individually by one of the co-authors, who was the knowledge engineer in this study. The remaining five salespeople were involved in the focus group session conducted by an experienced focus group facilitator and analyst. He, however, had not reviewed the book knowledge, nor had he been involved in any in-depth discussions concerning the sales domain. This situation emulates the "real world" where the knowledge engineer may be an inexperienced interviewer, but has spent weeks (maybe months) attempting to gain an understanding of the domain [Forsythe and Buchanan89]. The facilitator of the focus group, however, may not need this knowledge, due to the nature of the focus groups, i.e., the group interaction during the session.

4.4 Recording and Measurement

For both the individual interviews and the focus groups, the videotaped scenarios were presented and stopped at breaks representing the five segments. During the pause, the participants were asked to discuss their reactions to the scene; i.e., opinions, problem-solving approaches, and feelings. Both methodologies were recorded, and, in addition, the focus group session was videotaped.

All the audio tapes were transcribed and "coded" by identifying the following:

- those phrases that represent "book knowledge,"
- those phrases that pertain to solving the task at hand, but not noted in the trainee documentation, called "other knowledge," and
- those phrases relevant to all scenarios, i.e., general knowledge, or applicable only to the scenario being viewed, called "specific knowledge."

All the comments identified as "other" were highlighted and assessed by a panel of judges to determine the degree of newness of an idea, and, more importantly, its usefulness. The judges

were bank officials with greater experience and a higher organizational position than the participants. The judges were asked to evaluate the phrases identified as "other knowledge" along three lines: quality, acceptance, and originality. Each "other knowledge" phrase was subjectively ranked as high, medium, low, or not applicable (n/a) by the judges, based on written instructions. Two scoring sheets were needed for each segment within a scenario: one for the focus group and one for the individual responses.

4.5 Results

The purpose of the study was to see if there was a difference between the responses of the focus group and the individual interviewees in terms of "quality," "acceptance," and/or "originality," as scored by the three judges. The results showed that the focus group responses evaluated by the three judges were consistently rated higher than the responses elicited from the individual interviews. This was evidenced within each of the scenarios overall, and across all three evaluation criteria of quality, acceptance, and originality. Evaluation of segments within scenarios showed only five instances where the individuals rated a greater proportion of "high" valued responses than the focus groups.

For the overall proportional comparison, the focus group responses were rated as of "high" value in a range of 6 to 19% greater than the individual interviews. More significantly, when the "high" or "medium" proportions were added and compared, the focus group results still performed as well as, if not better than, the individual interviews.

Although not specifically examined, conflict resolution within the focus group was an additional benefit over using individual interviews. Several instances occurred among the individual interviews where approaches to the task were in direct opposition. For example, in the President's scenario, there was conflict between three of the interviewees as to whether they (as bank representatives) should indicate (to the client) if they could approve a loan or not. To supplement the Sales Training Manual and/or build rules for a decision aid, instances of conflict such as this (with the respective underlying lines of reasoning) must be resolved. Within the focus group session, however, when conflict arose, the group was able to probe each others' lines of reasoning. For example, in the Treasurer's scenario, there was disagreement on how to handle the clients' priorities versus the bank priorities. What initially started as opposing viewpoints was transformed, via group discussion, into an agreeable and innovative solution.

5.0 CONCLUSIONS AND DISCUSSION

This research compared the responses elicited from multiple experts in a focus group setting with individual interviews. The research vehicle, a sales call, can be described as an ill-structured or ill-defined problem. Simon [1960] describes such a problem as one with "no cut-and-dried" method of resolution since its nature and structure are elusive and complex. Mason and Mitroff [1983] further describe such a problem as one without criteria which could determine whether a solution was true or false; solutions may be deemed good, bad, or reasonable, but never true or false. The attempt was made to elicit knowledge from multiple experts to deal with the ill-defined sales situation.

One might propose that the beginning of the development process of an expert system is, in reality, an ill-structured problem. Recognition of a need and specification of a problem or decision situation are usually followed by a conceptualization process that includes postulating alternative views. Focus groups could be used to review preliminary specifications or definitions and generate new specifications, hypotheses, etc., based upon the insights of the participants. Our research showed that focus groups have value in generating original thoughts about a subject. Problems requiring convergent thinking may be analyzed and solved by technical ingenuity, while

divergent ones required more of creativity relevant skills. Studying creativity, Amabile [1983] found that to produce an original response requires domain relevant skills, i.e., talent, and innate cognitive abilities based on formal and informal training, i.e., experience. The focus group was found to produce responses of greater originality than the individuals with equivalent experience. This may be attributable to the group interaction.

In effective groups, members have been found to guide other members by asking appropriate questions, and pointing out important information [Hirokawa and Pace83]. This point is particularly relevant to the use of groups, specifically focus groups, as a means for knowledge acquisition. The members of the group can challenge each other, generating new ideas and resolving conflict.

After the problem is defined, viewing a focus group of experts would be helpful in orienting the knowledge engineer to the problem domain. The knowledge engineer would be better prepared to use either observation or interviewing to elicit knowledge from domain experts. The combination of book knowledge and viewing a focal group should also provide a range of thoughts, ideas, or guidelines which should help in making a preliminary assessment of the appropriateness of various knowledge representation schema.

6.0 NEXT STEPS

Building expert systems is a much richer process than that described by the elementary framework of [Hayes-Roth, et.al.83]. The three-step Identification, Conceptualization, and Formalization framework provides a starting point from which to consider the process of effective knowledge acquisition, but a more robust framework, such as that detailed in Figure 2, is infinitely more useful to researchers and practitioners. However, this larger framework needs to be empirically evaluated in order to provide sound guidance.

This research is our second step in the development of a broad knowledge acquisition research program. This research provided an initial framework for assessing the contribution of group knowledge acquisition activities to a suite of knowledge acquisition activities, defined in the context of the mixed knowledge acquisition methodology previously explored. Our next steps include a more robust assessment of the contribution of group vs. individual knowledge acquisition sessions (beyond the scope of focus groups), and a continuing synthesis of the contributions of different KA techniques to a well-developed KA process.

REFERENCES

[Advertising Research Foundation85]

Advertising Research Foundation, (1985), Focus Groups: Issues and Approaches, New York: Advertising Research Foundation.

[ALBM89]

DARPA/US Army Airland Battle Management program, (1989), FIRES-D Functional Requirements Report, June 1989.

[Amabile83]

Amabile, T.M., (1983), The Social Psychology of Creativity, New York: Springer-Verlag.

[Benbasat and Dhaliwal88]

Benbasat, I., and Dhaliwal, J.S., (1988), "A Framework for the Validation of Knowledge Acquisition," Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop," AAAI, Banff, pp. 1.1 - 1.18.

[Boose88]

Boose, J.H., (1988), "A Survey of Knowledge Acquisition Techniques and Tools," Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop, AAAI, Banff, pp. 3.1 - 3.23.

[Burton et.al.87]

Burton, A.M., Shadbolt, N.R., Hedgecock, A.P., and Rugg, G., (1987), "A Formal Evaluation of Knowledge Elicitation Techniques for Expert Systems: Domain 1," Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems," Reading University, pp. D3.1 - D3.20.

[Collins and Guetzkow64]

Collins, B.E., and H. Guetzkow, (1964), A Social Psychology of Group Processes for Decision-Making, New York: John H. Wiley, Inc.

[Cooke and McDonald86]

Cooke, N.M. and McDonald, J.E., (1986), "A Formal Methodology for Acquiring and Representing Expert Knowledge," Proceedings of the IEEE, 74:10, pp. 1422-1430.

[Davis83]

Davis, W.S., (1983), Structured Systems Analysis and Design, Reading, Massachusetts: Addison-Wesley Publishing Company.

[Davis and Olson85]

Davis, G.B. and Olson, M.H., (1985), Management Information Systems, 2nd edition, New York: McGraw-Hill Book Co.

[Dhaliwal and Benbasat89]

Dhaliwal, J.S. and Benbasat, I., (1989), "A Framework for the Comparative Evaluation of Knowledge Acquisition Tools and Techniques," CORS/ORSA/TIMS Conference, Vancouver, BC.

[Ericsson and Simon84]

Ericsson, K.A., and Simon, H.A., (1984), Protocol Analysis: Verbal Reports as Data, Cambridge, Massachusetts: The MIT Press.

[Fern82]

Fern, E.F., (1982), "The Use of Focus Groups for Idea Generation: The Effects of Group Size, Acquaintanceship, and Moderator on Response Quality and Quantity," Journal of Marketing Research, 19, pp. 1-13.

[Fern83]

Fern, E.F., (1983), "Focus Groups: A Review of Some Contradictory Evidence, Implications, and Suggestions for Future Research," Advances in Consumer Research, 10, pp. 121-126.

[Fisher74]

Fisher, B.A., (1974), Small Group Decision-Making, New York: McGraw-Hill.

[Forsythe and Buchanan89]

Forsythe, D., and Buchanan, B., (1989), "Guidelines for Knowledge Acquisition," IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on Knowledge Acquisition, 19:3.

[Goldman62]

Goldman, A.E., (1962), "The Group Depth Interview," Journal of Marketing, 26, pp. 61-68.

[Grabowski87]

Grabowski, M.R., (1987), "Prototyping As a Design Strategy and a Means of Knowledge Acquisition for Expert Systems: An Assessment Using 'The Piloting Expert System'", Ph.D. Thesis, Rensselaer Polytechnic Institute, Troy, New York.

[Grabowski90]

Grabowski, M.R., (1990), "A Mixed Knowledge Acquisition Methodology Reconsidered," Department of Decision Sciences and Engineering Systems, Technical Report 90-P-145, Rensselaer Polytechnic Institute.

[Grover83]

Grover, M.D., (1983), "A Pragmatic Knowledge Acquisition Methodology," Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83), Karlsruhe, West Germany, pp. 436-438.

[Hare88]

Hare, A.P., (1988), Creativity in Small Groups, Sage, California: Sage Publishing.

[Harrison87]

Harrison, E., (1987), The Managerial Decision-Making Process, Boston: Houghton Mifflin Publishing Company.

[Hayes-Roth, et. al.,83]

Hayes-Roth, F., Waterman, D.A., and Lenat, D.B., (1983), "An Overview of Expert Systems," in Hayes-Roth, Waterman and Lenat (eds.), Building Expert Systems, Reading, Massachusetts: Addison-Wesley Publishing Company, pp. 3-29.

[Hess68]

Hess, J.M., (1968), "Group Interviewing," in 1968 ACR Conference Proceedings, R.L. King (ed.), American Marketing Association, IL.

[Hirokawa and Pace83]

Hirokawa, R., and Pace, R., (1983), "A Descriptive Investigation of the Possible Communication-Based Reasons for Effective and Ineffective Group Decision-Making," Communications Monographs, 50, pp. 363-80.

[Hoffman87]

Hoffman, R.H., (1987), "The Problems of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology," AI Magazine, 8:2, pp. 53-67.

[Kitto88]

Kitto, C.M., (1988), "Progress in Automated Knowledge Acquisition Tools: How Close Are We to Replacing the Knowledge Engineer," Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, pp. 14.1 - 14.13.

[Leifer, Durgee and Wallace89]

Leifer, R., Durgee, J., and W.A. Wallace, (1989), "Acquiring Deep Knowledge: A New Approach," 1989 CORS/TIMS/ORSA Joint National Meeting, Vancouver, British Columbia, May.

[Linster88]

Linster, M., (1988), "A Critical Look at KRITON: Moving From First to Second Generation," Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop, AAAI, Banff, pp. 17.1 - 17.19.

[Mason and Mitroff83]

Mason, R.O., and I.I. Mitroff, (1983), "A Program for Research on Management Information Systems," Management Science, 19, 1983, pp. 475-487.

[Massey and Wallace90]

Massey, A.P. and Wallace, W.A., (1990), "Focus Groups as a Knowledge Acquisition Technique: An Exploratory Study," Department of Decision Sciences and Engineering Systems, Technical Report.

[Merton, Fiske and Kendall56]

Merton, R.K., Fiske, M., and Kendall, P.L., (1956), The Focused Interview, Glencoe, Illinois: The Free Press.

[Morgan88]

Morgan, D.L., (1988), Focus Groups as Qualitative Research, Sage University Paper Series on Qualitative Research Methods, Sage, California, 16.

[Newell and Simon72]

Newell, A. and Simon, H.A., (1972), Human Problem Solving, Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

[Prerau87]

Prerau, D.S., (1987), "Selection of an Appropriate Domain for an Expert System," AI Magazine, 6:2, pp. 26-30.

[Reitman Olson and Reuter87]

Reitman Olson, J., and Reuter, H.H., (1987), "Extracting Expertise from Experts: Methods for Knowledge Acquisition," Expert Systems, 3:3.

[Senn89]

Senn, J., (1989), Analysis and Design of Information Systems, 2nd edition, New York: McGraw-Hill, Inc.

[Shaw and Woodward88]

Shaw, M.L.G., and Woodward, J.B., (1988), "Validation in Knowledge Support System: Replication and Consistency with Multiple Experts," International Journal of Man-Machine Studies: Special Issue on Knowledge Acquisition for Knowledge-Based Systems.

[Simon60]

Simon, H.A., (1960), The New Science of Management, New York: Harper and Row.

[Weitzel and Kershberg89]

Weitzel, J.R., and L. Kershberg, (1989), "Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle," Communications of the ACM, 32:4, pp. 482-488.

[Zmud88]

Zmud, R.W., (1988), "Specification Structure and Measurement Strategies for the NonRepeating Group Decision Context," in Organizational Decision Support Systems, R.M. Lee, A.M. McCosh, and P. Migliarese (eds.), Amsterdam: Elsevier Science Publishers.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

The Use of Lotus Agenda in Knowledge Acquisition for an Expert System in a Poorly Controlled Domain

Anton R. Grashion
Rocky Mountain Banknote
10455 W. 6th Avenue
Lakewood
Colorado 80215
USA

Abstract

Knowledge elicitation is perceived traditionally as a bottle-neck in the production of commercial expert systems. Nowhere is this more apparent in areas where there is a scarcity of available expertise or opposition to new technologies. There is a paradox that exists in some domains suitable for expert system development where, due to the very scarcity of expertise, the current experts are so overworked that they cannot be spared for other tasks. This situation also runs counter to the laws of supply and demand, where an increasing demand for expertise is not met by supply. However, where a body of published information is available to a non-naive knowledge engineer there are newly available software tools that simplify the structuring of information into an intermediate knowledge representation from free-format text.

1. INTRODUCTION

Whether one agrees with the traditional perception that there exists a shortfall between potential expert systems and the acquisition of knowledge for such systems, depends to some extent upon what we believe the process of knowledge elicitation entails and what results it can be expected to produce. If, for a given field of study, we accept that knowledge elicitation is unlikely to lead to a complete and correct 'knowledge-set'(Young, 1988) then it may be helpful to relax our expectations of a potential system, restricting the more fanciful claims of expert behaviour while concentrating on the manipulation of the acquired knowledge of the domain. Any perceived 'bottle-neck' may also be alleviated by devolving expert system development to the users. This goal may be achieved with the development of 'friendlier' AI programming environments running on affordable hardware of a high specification allowing the development of expert by the expert users themselves. This of course relies heavily upon the motivation of the experts and the time available to them.

The area in which I am working currently has several problems associated with knowledge elicitation. Firstly, Although collaborating with experts in the domain their time is severely

restricted due to commercial commitments. This problem is somewhat counteracted by the fact that I am conversant with the domain in question, that of environmental analysis of sedimentary facies. Secondly, there is marked reticence on the part of the geologists towards the use of 'expert' systems. This is in part due to over-extravagant claims of expert behaviour during an earlier project. Thirdly, in an attempt to improve the geological accountability of the system the adoption of a causal representation of the domain was proposed. This leads us into very murky and poorly controlled areas where agreement between experts is rare. For this reason sedimentary facies analysis would be generally be considered a poor candidate for an AI based system.

The double problem of having to formulate the domain frames of reference as well as providing the knowledge representation mapping into some implementation (Kidd and Sharpe 1988) presents many problems. In response to these problems the approach advocated in this paper relies heavily upon the fact that the knowledge engineer is non-naive with regard to the problem domain. This requirement is realised in a growing number of projects as improved interfaces to complicated shells and tools enable experts or expert teams to construct intelligent knowledge based systems 'in house'. It also means that the burden of solving two problems (domain and representation) lies with the user/developer. However the method has the advantage that systems developed in this holistic manner would tend to result in a co-operative problem solving approach with the user being left to deal with those aspects of problem solving that they are adept at (i.e., visual perception), and the system contributing to the symbolic reasoning and information gathering tasks it is suited to.

2. INTERMEDIATE KNOWLEDGE REPRESENTATIONS.

An ideal intermediate knowledge representation (Young, 1988) in the case of geological environmental analysis would be the drafted core-log, constructed by geologists after their examination of a cored-section. However, this would require a prohibitively (in terms of the aims of the project) large graphical programming effort and therefore the lithofacies comprising the core-log are instead reduced to alphanumeric ciphers in what is termed section standard form (SSF) to simplify input (Grashion 1988). The next best intermediate knowledge representation would be that of free-form text describing the core, its components, and their relationships. Geologists, during their work, make copious annotations of core-logs and of course there exists a large, and growing, corpus of research information contained within various technical journals. To satisfy the self-documenting role of an intermediate knowledge representation it is desirable that the knowledge that it represents be of a readily digestible form. Mere collation of relevant papers in an ad hoc fashion would serve only to obscure an eventual system rather than illuminate it. In addition, it is evident that

technical papers do not actually contain 'expertise'. Although the precise nature of expertise is contentious, clearly it is more of a skill than a commodity, and is the process of interpretation and application of knowledge in a flexible and efficient manner. Consequently, it seems that some form of pre-processing of textual information is required prior to ordering and presentation of the data as an intermediate knowledge representation.

The pre-processing stage is accomplished by the knowledge engineer or developer reading relevant papers and extracting short, concise and meaningful phrases, an activity regularly performed when perusing technical articles. Obviously this technique can be applied to the traditional knowledge acquisition tasks of analysing interview transcripts and written protocols. The knowledge engineer's skill at this stage is the ability to distil 'items' relevant to the system under development from the technical documentation. The results of this process are a haphazard collection of noted facts, heuristics, guesses, hypotheses - in fact a conglomerate of information fragments that partly characterise the domain in question in a structureless fashion. Not only is this melange of data structureless, but most likely contains inconsistencies and contradictions, especially in the likely event that the information has been derived from more than one source. Evidently the imposition of a structure, or rather the discovery of an existing, but obscured, underlying structure of the information, would provide an initial mapping onto a knowledge representation. It is this process of structural discovery which personal information management systems can afford developers in a relatively compressed time-scale.

3. PERSONAL INFORMATION MANAGEMENT SYSTEMS

This rather grandiose title has been conferred on a number of commercially available programs. The software can be divided broadly into two categories. The first category contains those programs designed to maintain a free-format text-base and allow some structural links to be created and supported within it. Programs in this category include Hypertext systems, Micrologic's Tornado, PC-Outline, Grandview, Persoft's IZE and more recently GEM based WEB. Programs in the second category have a similar specification with additional features concerned with times, dates and appointments. The most notable example of software in the second category is Lotus Agenda. The purpose of this article is to present an example of how information can be collected, structured and analysed by using Agenda for knowledge acquisition purposes. However, the same functionality could be achieved by combining several individual programs.

4. AGENDA IN THE KNOWLEDGE ACQUISITION PROCESS

The process of using Agenda to aid the knowledge acquisition task breaks down into four steps:

1. Prepare text
2. Import text into Agenda
3. Add new items
4. Construct Categories
5. Create views of text-base

Steps 3 to 5 can be iterated until the developer is satisfied that he has explored all fruitful implications of the text-base. In fact the above steps can all be repeated during the course of the knowledge acquisition process as new information is obtained.

4.1 Preparation of Text

During a directed reading program for research of any kind, a collection of random notes pertaining to the project will be amassed. I make the assumption that the notes have not been directly entered into Agenda (though this would be desirable). The collected notes may impinge upon all aspects of a project, from programming ideas to extracts from reference material. The collection of material may extend across a number of media including scraps of paper and word-processed files. At a simple, organisational level, Agenda can provide a single coherent medium for recording variable length textual information about a project.

There are a few concepts that we must familiarise ourselves with before we may enter a detailed discussion of Agenda's features. There are three basic components of the Agenda system; items, notes, and categories. Items are, not surprisingly, items of information that may be up to 350 characters in length and comprise the fundamental input to Agenda. The size restriction is often criticised in review articles, and if one simply compared Agenda with a relational database for free-format text it would indeed be a meagre figure. However, as I hope to point out later this limit forces us into a discipline of brevity that will be repaid when structuring the text-base. Notes are attached to items to provide greater detail and to expand the 350 character limit. Notes are either held in the current text-base, or as separate files on disk, in either case each note may not exceed 10k in length. Categories provide a hierarchical structure by which we may classify items in the text-base.

4.2 Importing Text

This stage of preparation requires that collected information, from outliner, word-processor etc., be saved as an ASCII file with blank lines delimiting each item. The ASCII file is then processed by a utility program, TXT2STF, supplied with Agenda

that translates the text file into a structured file. The TXT2STF utility produces a file with a .STF extension which, in its simplest form contains the original text with embedded annotations indicating where items begin and end. The resulting structured file may then be imported into the Agenda program using the File-Transfer-Import option.

The initial category hierarchy contains only two categories. MAIN is the default parent of the hierarchy and inherits all items from its children. The category 'Untitled' was specified during the import operation and has had all imported items assigned to it during this process. During construction of the category hierarchy, the items of the text-base will be assigned to categories either automatically by Agenda, or directly by the developer. Figure 1 shows the imported information displayed in the Initial View of Agenda. Views in Agenda are the means by which the text-base of items and their associated categories can be examined. The Initial View is a default showing all information items under the default section heading of category

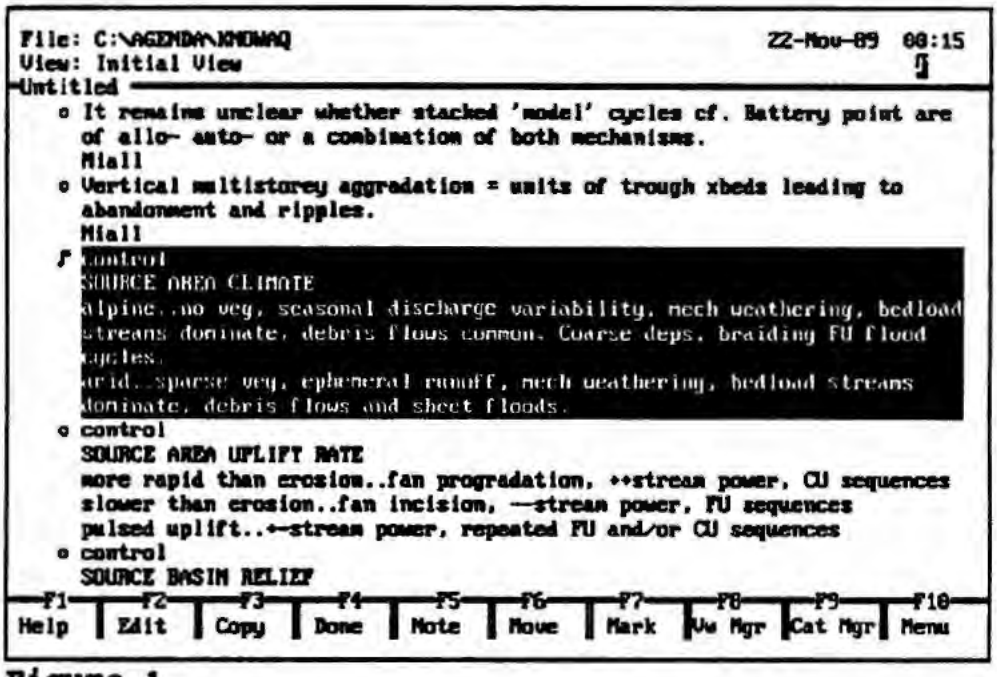


Figure 1
Initial view following structured file import

Untitled. All section headings in a view must be categories, and all items assigned to the category of the section head are displayed below it.

After importing a structured file it is prudent to scan the text-base to determine how items, exceeding the 350 character limit, have been divided into items and notes. Figure 1 and Figure 2

show an item divided into item and attached note. On examination it would appear that the item originally contained too much detail. A solution is to create new items from the note text as shown in Figure 3. Following this tuning operation we can start to expand the category hierarchy.

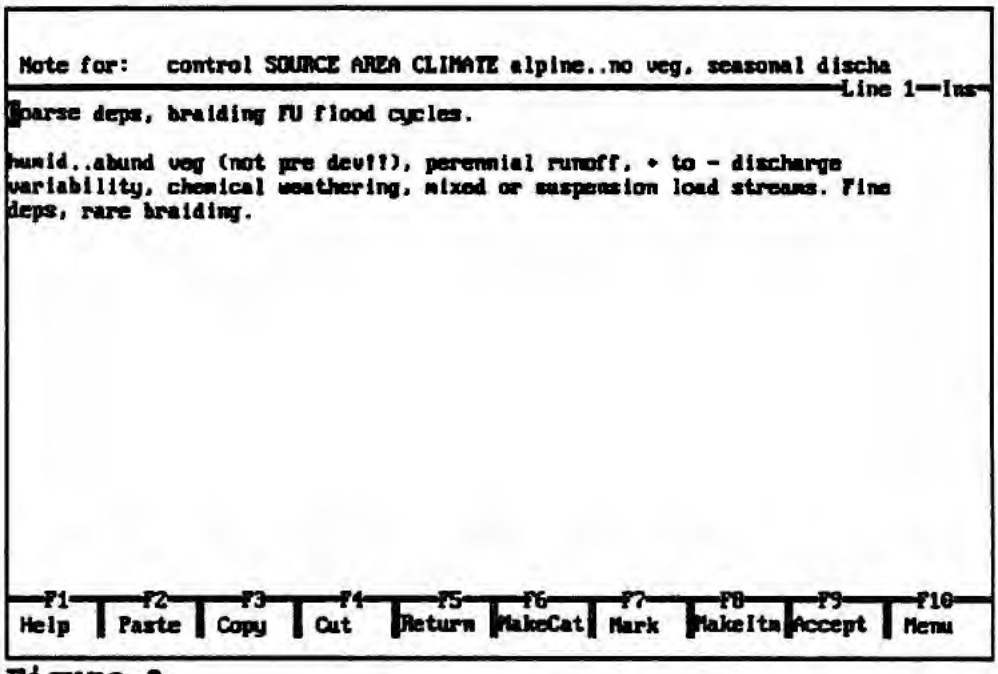


Figure 2

4.3 The Category Hierarchy

In its simplest form, an individual category may be thought of as a 'pigeonhole' into which items in the text-base are assigned. Individual categories may be defined as names, single words, numbers or phrases up to 67 characters long with no limit imposed on the number of categories. Categories are related by a hierarchical structure with MAIN as the initial overall parent of the hierarchy and "Untitled" its only child. These category names may be edited to more meaningful names or phrases if the developer wishes. Assignment of items to categories can performed automatically using the Utility-Execute menu selection, or manually using the Assignment-Profile option. In either case, an initial category hierarchy must be created.

There are two basic techniques of creating a category. The first, working in View mode, is to add a new section or column to the current view. A section is defined in a view by specifying a category name (in this case a new one) that is then displayed as

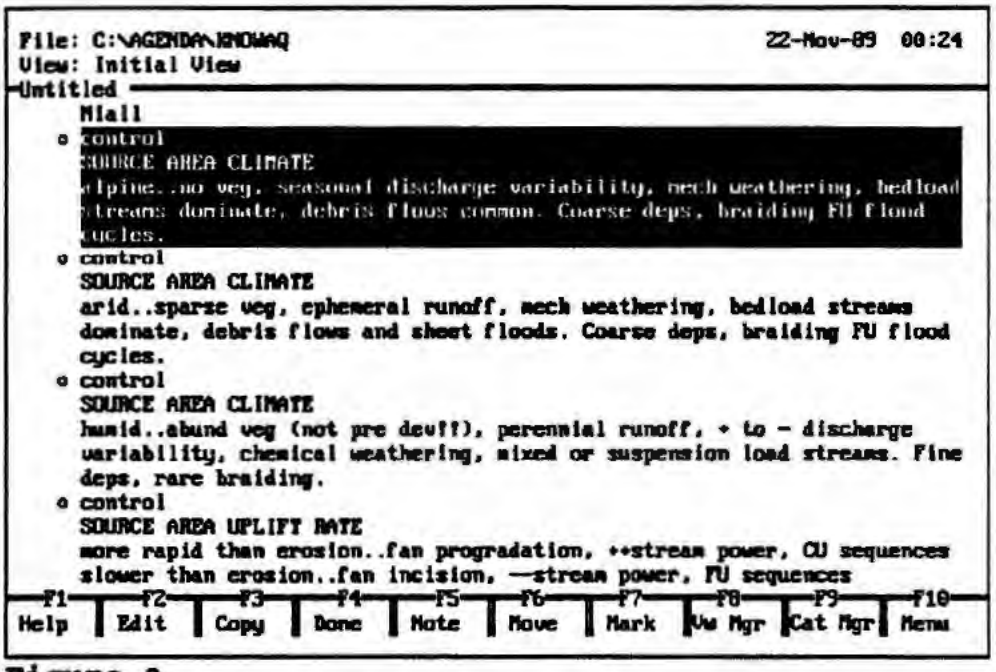


Figure 3
New Items created from note in Figure 2

a heading for all the items assigned to it. Any item later addedunder this section heading are assigned to the section heading category. The second method, and the one most suitable to our current example is to add the categories via the category manager screen. This method is particularly applicable where we have some knowledge of the underlying structure of the text-base.

A category hierarchy is extremely flexible and can be modified without loss of data or complicated restructuring at any time. In fact, altering categories and their relationships 'on the fly' is a positive advantage over traditional databases and allows us to experiment and view the text-base from many different perspectives.

Figure 4 shows a fragment of the category hierarchy used for the current project and illustrates some important points. The parent category 'cyclicity' also has a synonym of 'cycles'. Therefore, when checking the text-base during automatic assignments, items containing the word "cyclicity" OR the word "cycles" will be assigned to the category. The category has one child category with two synonyms. Any item assigned to the 'allo-;auto-;modal' category will also be assigned to the parent category 'cyclicity;cycles' by upward (reverse?) inheritance. Logical AND is effected by separating the synonyms by a comma instead of a semi-colon. The next category has a synonym that is most unlikely to be encountered in any input text, namely 'sed-stru'. The reason for this is that as the primary category entry ('sedimentary structures') is so long, it would be severely truncated if used as a column heading in a view. When this sort

of situation occurs Agenda uses the first synonym as the column heading if one is available and if it is shorter. Therefore this synonym has been added simply as an aid to readability when using views. The next parent 'lithofacies' has children that are mutually exclusive. Any item assigned to one of these children cannot be assigned to any other of the same family.

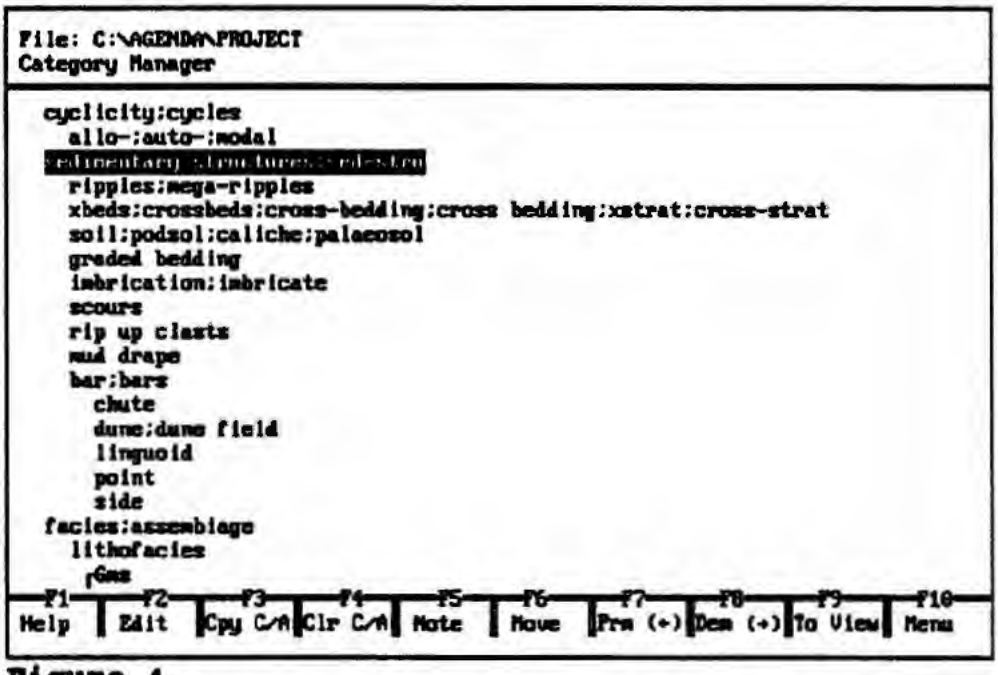


Figure 4
Category hierarchy fragment

4.3.1 Auto-Assignment

As mentioned in section 4.3, assigning items to categories can be achieved in a number of ways. It is the flexibility of the auto-assignment facility that relieves the developer of this otherwise time-consuming chore. New items added to the text-base are automatically scanned to determine whether any of the new text matches the literal synonyms of the categories. This auto-assignment can be forced at any time, for instance if new synonyms are added or the hierarchy restructured, by means of the Utility-Execute menu choice. Figure 5 shows the auto-assignment preference screen illustrating the flexibility of this operation. Disabling text conditions effectively turns the pattern-matching feature off. The next option allows determination of whether item and note text, item only, or note only will be matched against the categories. This means that balance between separate items and items with associated notes is, to some extent, subjective, as category assignments can be made to operate on both types of text. However, dividing items into short, logical and coherent points prevent items with notes being assigned to multiple disparate categories. In fact when this situation arises a review of item size should be undertaken. Required match strength can be exact, partial or minimal. An exact match requires all the words

in the category phrase to be in the item. A partial match is achieved if the item attains a match strength of 50%. A minimal match requires only 1% (or one word) to be present in the item. Single word categories therefore always provide exact matches. Confirm assignments allows the developer to chose to be informed of assignments and to arbitrate between possibly ambiguous assignments. The remaining options are self-explanatory. Besides these preference settings, the category hierarchy can be adjusted to facilitate the required matching. Wildcards * and ? have the same usage as in DOS and can be placed in category text, as well as attaching assignment profile conditions and actions.

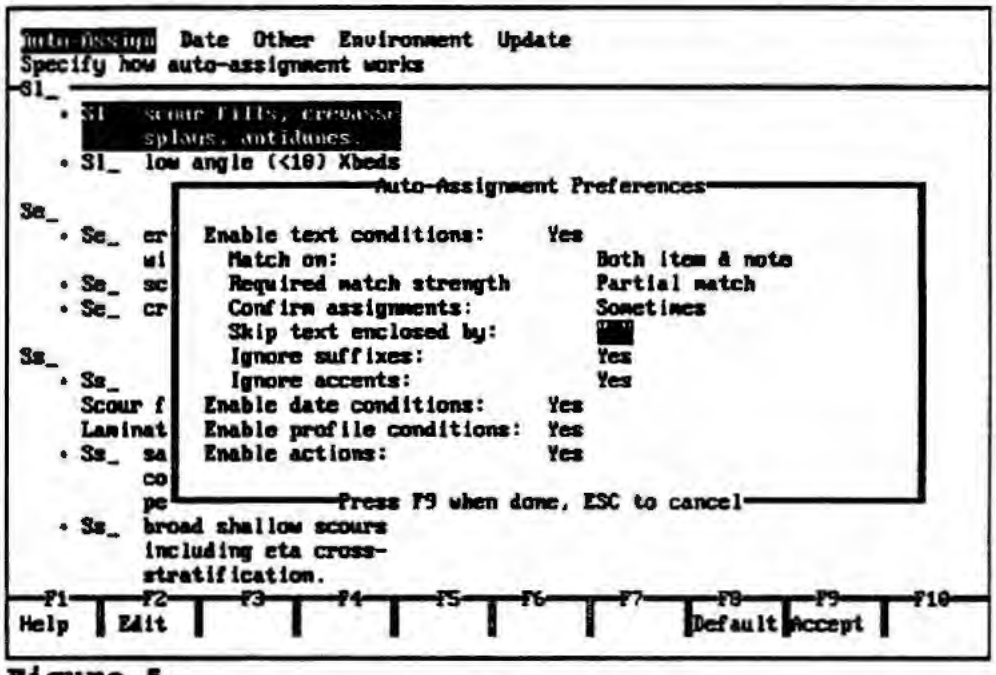


Figure 5
Auto-assignment preference settings

Figure 6 shows the category hierarchy with some sub-hierarchies collapsed. This figure also illustrates categories with attached conditions. When items are assigned to categories via the auto-assignment facility, Agenda checks the hierarchy for profile conditions. These conditions assign an item matching some other categories to the category that has the condition attached. In Figure 7 any item not assigned to a category is assigned to the 'unassigned' category (otherwise we could not view such items). This enables us to locate those items that our current hierarchy does not cover, thereby indicating the need for extra categories or deletion of items. The NEXPERT OBJECT sub-hierarchy also contains a category with a profile condition. Items assigned to the category 'control' will also be assigned to the 'Causal' category. Profile actions are similar to conditions but even more flexible. They can assign an item to many categories also enabling special actions to be taken such as exporting items to other files or discarding unwanted items.

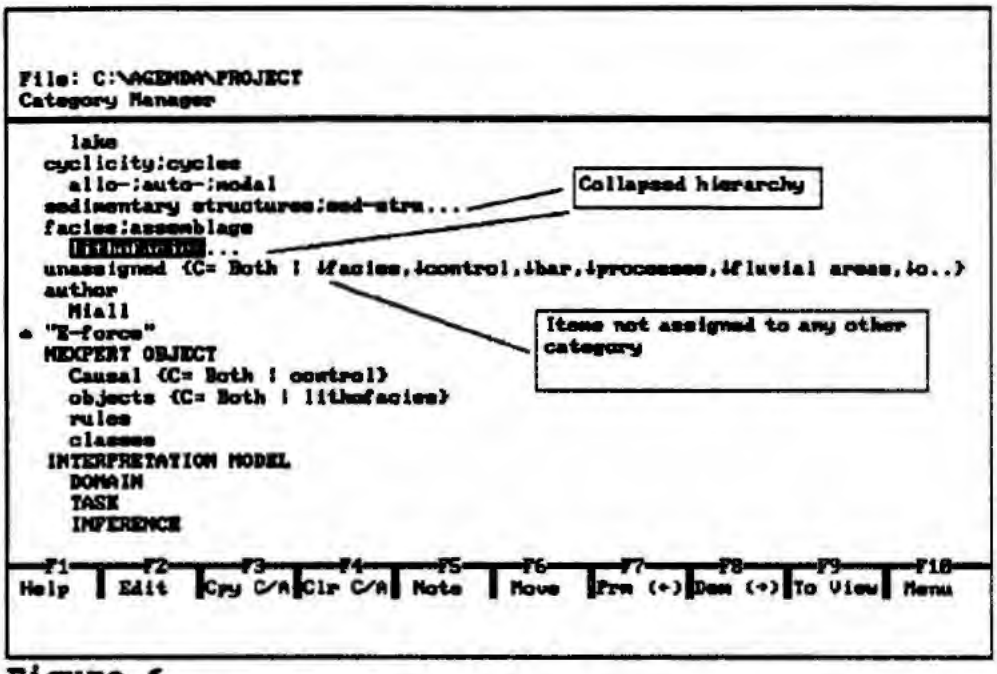


Figure 6
Category hierarchy with collapsed families

Viewing the results of assignments, auto or manual, is a straightforward matter. The simplest approach is to obtain the assignment profile for an item highlighted in a view by pressing the grey+ key or ALT F9. Figure 7 shows the assignment profile for an item placed in the soils section of the initial view following assignment. The category hierarchy is shown indicating which categories the item has been initially assigned to. These assignments can be altered directly from this screen and indeed new categories may be added. Assignment profiles can be displayed for a group of items by marking each item (F7). In this type of profile display, assignments are indicated by a number rather than an asterisk, identifying the which marked item the assignment is for.

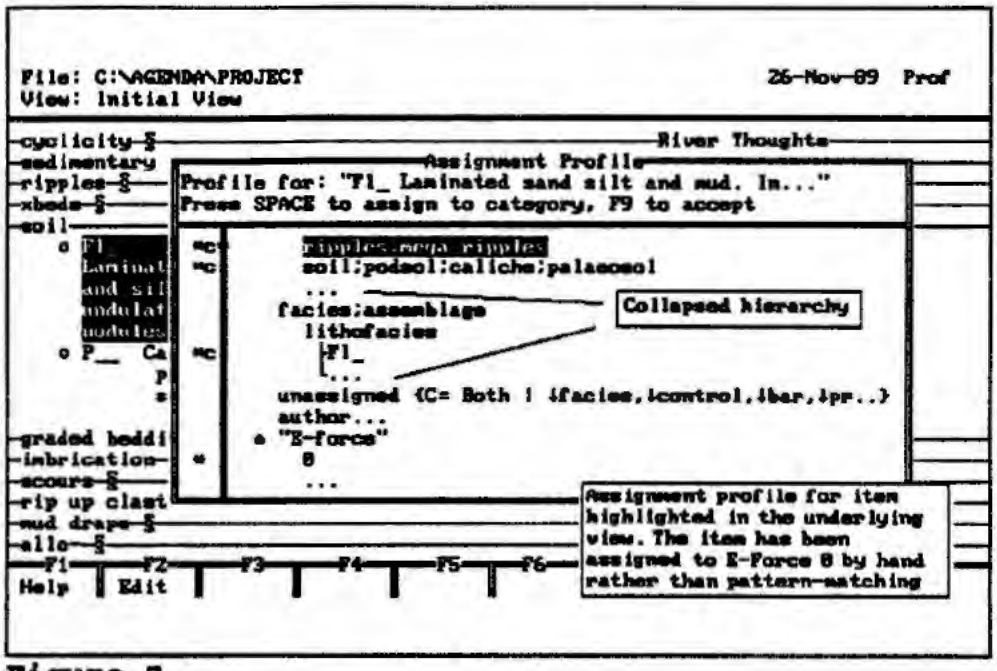


Figure 7
Assignment profile

5. CREATING AND USING VIEWS

The initial View becomes confusing and cluttered as more items are added. It is perhaps a fundamental strength of Agenda that multiple views can be created quickly and easily, affording the developer various perspectives of his data. Figure 8 shows a new view being created using category "control" as the parent category for the new view. Note also the auto-completion facility of agenda identifying the required category after only 2 keystrokes. The new view will comprise all the items assigned to the sub-hierarchy headed by category "control". This view also will contain items under sections headed by each of the child categories of "control". The view can be modified be applying filtering conditions to sections, re-ordering the sections, sorting items within the view, selectively collapsing certain sections or adding new sections at any time. Figure 9 displays the result of creating the new view 'By control' and displays all the items assigned to the "control" category family under their respective section headings. Only "climate" and "relief" sections from the "control" family can be seen on the screen in this figure.

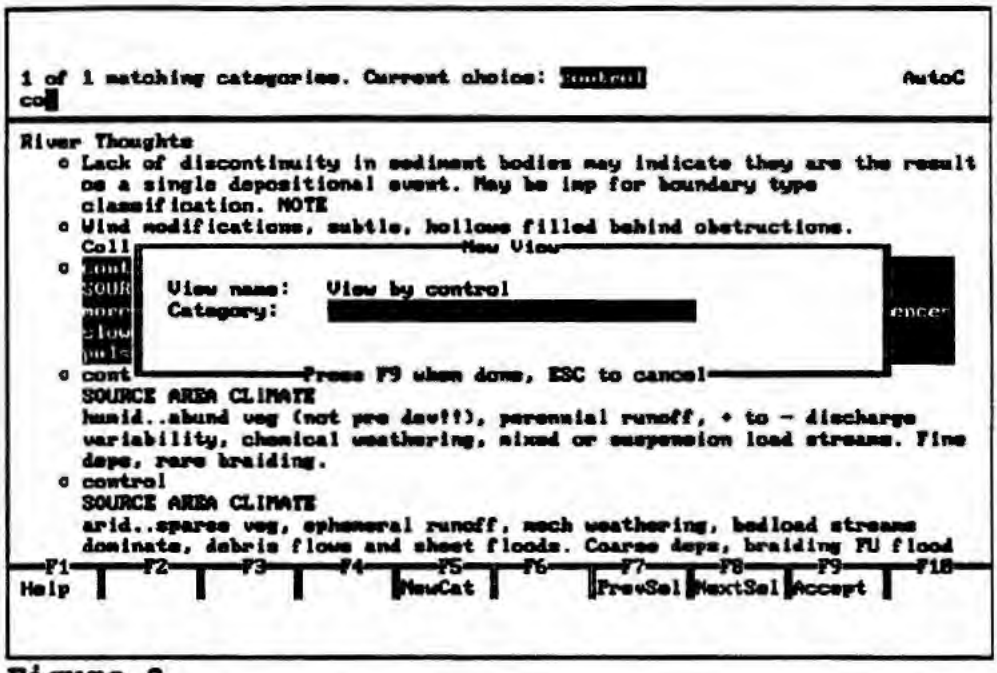


Figure 8
Creating a new view

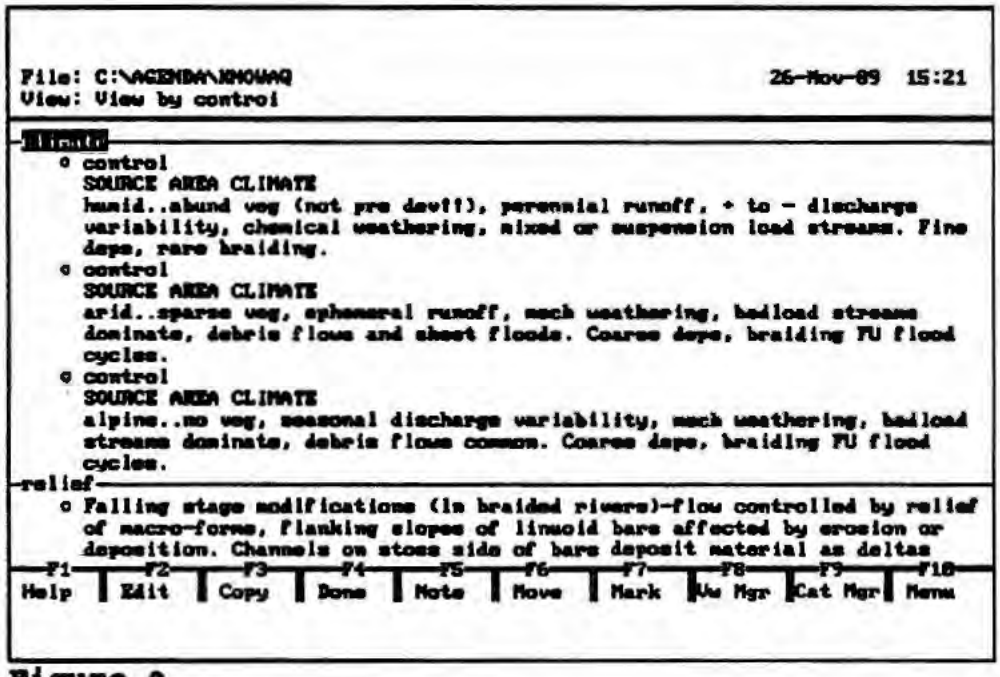


Figure 9
New view 'By Control'

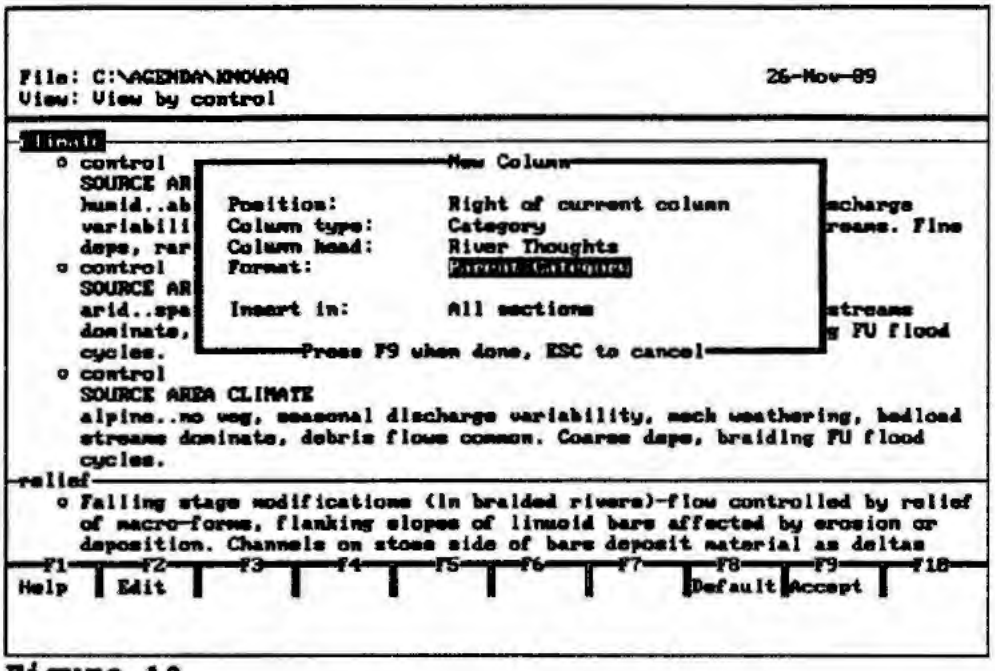


Figure 10
Adding a new column to a view

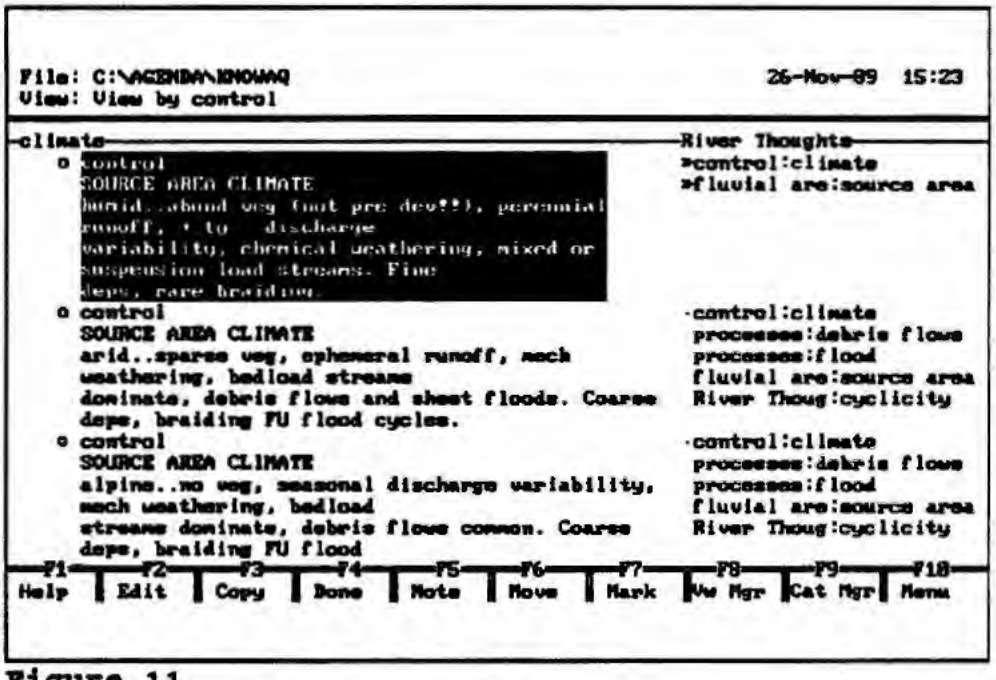


Figure 11
Result of adding the new column

More information can be squeezed into a view by adding columns headed by other categories. In this manner we can see how the items in the current view have been assigned to other members of the category hierarchy. Figure 10 illustrates the process of adding a new column in Parent:Category format headed by River Thoughts in all sections of the current view. As this is the overall parent category it provides us with a short-cut to displaying all the assignments for any item shown in a particular view. Obviously, if this column is added to the initial view, all assignments can be examined as all items are included. Various subtle formatting techniques may be applied to the columns to create the desired emphasis.

Figure 11 shows the view completed by the addition of the new column. Examination of the data in this column shows that the highlighted item is also assigned to the "fluvial areas" parent category via its child category "source area". Hence, this column provides similar information to that of the assignment profile as shown in Figure 7.

It should be clear by now that Agenda provides the developer with almost bewildering flexibility when ordering and viewing data. In fact, many end-results can be achieved via subtly different routes in Agenda that can be very confusing until enough time and effort is expended in coming to terms with the software.

5.1 Representation and Implementation Aspects

I believe that the flexibility that Agenda offers in its data viewing aspect that makes it particularly well suited to a rapid prototyping approach to knowledge acquisition. This viewing flexibility has a cost in other areas of the system, notably in relating items to each other and maintaining pseudo-relational links. Fortunately there exist some suitable work-arounds that can enable us to at least simulate dependence. Rapid prototyping, once the accepted method for expert system development, has been subject to criticism (Breuka and Wielinga 1988) by leading exponents of knowledge acquisition in deference to more formalised and rigorous techniques. I intend to show that Agenda can be used simultaneously for prototyping of the most unstructured kind and knowledge representation and documentation of a more formal variety.

5.1.1 Exploring Relationships

Although knowledge acquisition can, and perhaps should, proceed to a large extent in ignorance of implementational aspects of an eventual system. Rapid-prototyping would seem to require an awareness of the underlying structures available to the developer. In particular, it may be possible to work out representational issues using a simplified model of what will be implemented. For instance, the FLEAS project, from which this

paper results, uses the Nexpert Object expert system tool-kit. At an early stage of development the very complex representation formats available in Nexpert can be abstracted into a notional generic frame-based system. This system is made up of hierarchies of classes, sub-classes, objects, sub-objects and properties. By using the views of Agenda the developer can explore relationships between the various elements and from these infer both organizational structures in the object world as well as first-pass rules relating objects and object classes. As a simple illustration of this point consider the screen shot portrayed in Figure 12.

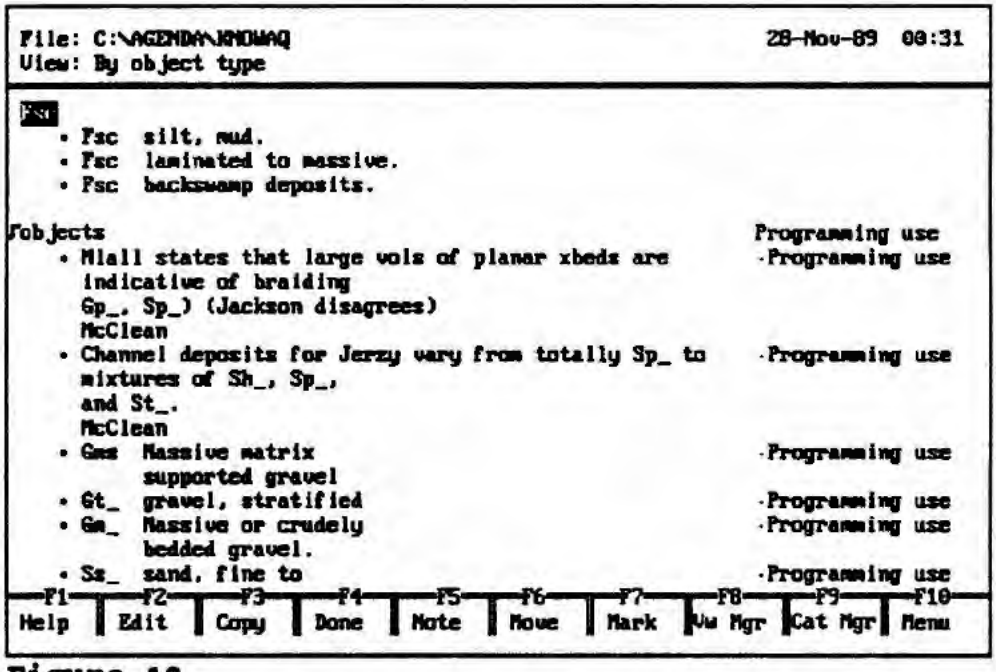


Figure 12
View by Object type with new section added

This shows a view 'By Object type' that the developer has added "Fsc" as a new section to focus on this object (object in implementational terms). This section provides the developer with summary information regarding Fsc. At this point the knowledge engineer may be interested in which other items are assigned to the 'backswamp' category as mentioned in the third item in the Fsc section. To explore any links between the Fsc object and other objects related to what may eventually become a backswamp object, the developer adds another section as shown in Figure 13.

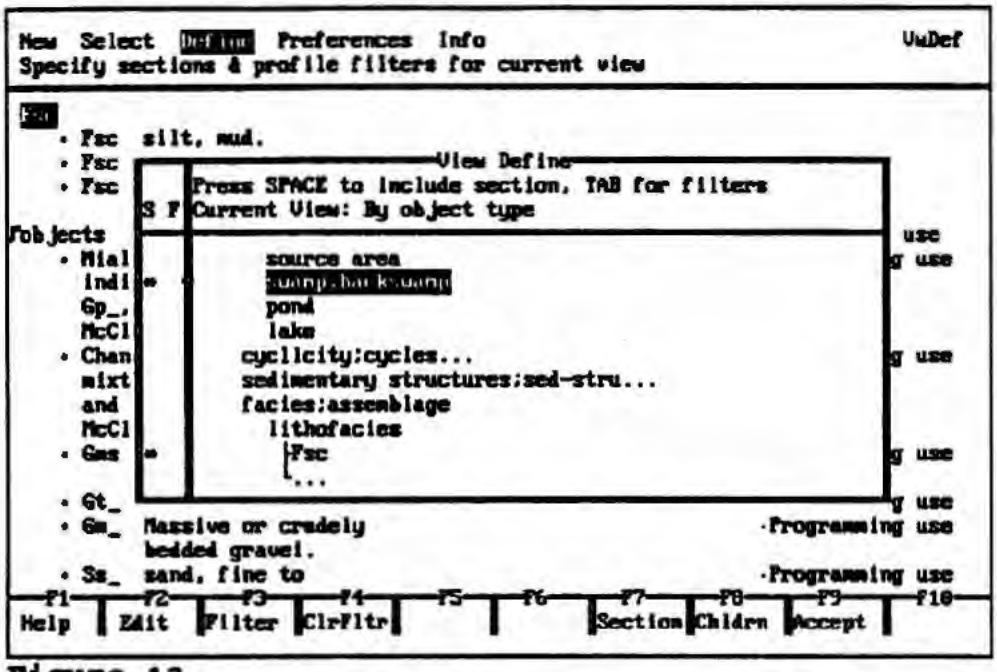


Figure 13
Including a new section in the view

This results in a redefinition of the view (as indicated by VwDef in the upper right-hand corner of Figure 14) and the new section is selected as the swamp;backswamp category. The result is displayed in Figure 14. The new section shows other items assigned to the swamp;backswamp category thereby highlighting relationships between these items and Fsc in the context of "swamp;backswamp" (note the relevant Fsc item is refiled under this new section as well). The results of this analysis can now be mapped directly into Nexpert's object hierarchy by creating classes, objects and links in the shell. Figure 15 is a screen-shot from Nexpert object showing illustrating this point. In this fragment of object hierarchy LITHOFACIES and OVERBANK_CLASS are parent classes. LITHOFACIES has several sub-classes (denoted by a _SC extension) determined by grain-size, to which various objects are attached. Fsc is an object belonging to the sub-class of fine_SC with Fm_ and Fcf. Fsc is also classified as an object under the sub-class of backswamp_SC that is a sub-class of OVERBANK_CLASS.

Adding a new column at this stage to determine where else these items are categorised may expand the investigation of relationships between the objects be delineating further associations. The actual semantics of relationships identified in this way can be traced to objects at many different levels in the system, from the domain object level up to strategic planning levels. Any interpretation placed on the relationship depends to a large extent as to what type of object the item generates. If the item generates an object associated with accomplishing some

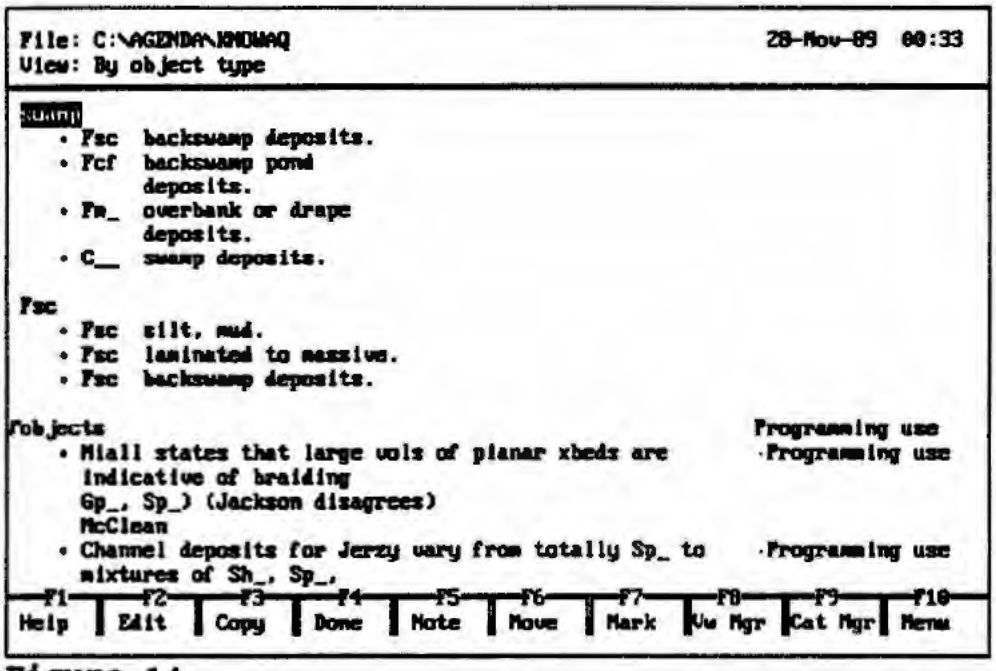


Figure 14

goal in the system, then the relationship is affected at the strategic planning or meta-rule level. It is evident that an iteration of this process will eventually uncover the comparatively obvious relationships between the items at all levels of abstraction.

The process of uncovering object associativity also may display areas of contradiction in the items, again at all levels of abstraction. Such discrepancies may be the result of disagreement between knowledge sources, inconsistent interpretation of information during text extraction, or inappropriate auto-assignment in Agenda. The latter case is easily checked by examining the assignment profile for the ambiguous items and making adjustments where necessary. The other two sources of incongruity require reinspection of the original documents from which the text items have been extracted. For this reason it may be wise to include some form of tag to the source of each item.

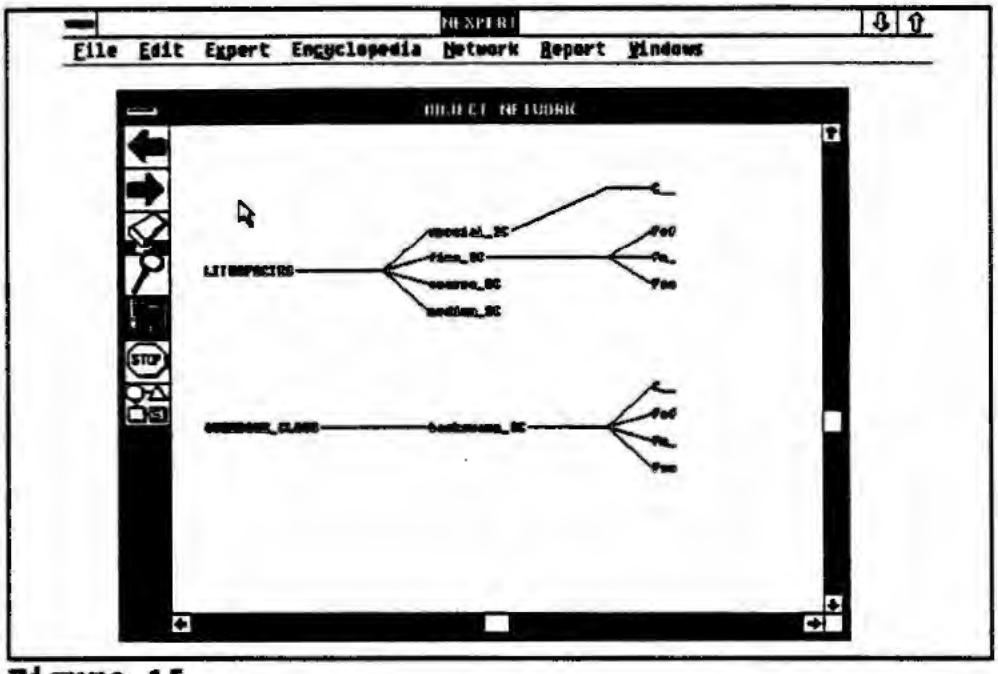


Figure 15
Nexpert Object screen

This tag may take several forms. Firstly, a reference to the original paper could be included as a note for each item. This method would lead to an unnecessary duplication of data if many items are gleaned from the same paper. A solution to this would be to attach a disk file, containing the reference as a note of the items. The same file could be attached to as many items as appropriate. However, this could lead to a large number of text files cluttering up the hard disk. A more elegant solution would be to include the actual reference as part of the category hierarchy. The simplest method of implementing such a mechanism is to make the title of the reference an item. This item, should be assigned to a family of unindexed categories headed by a REFERENCE category (unindexing hides the children of such categories). Addition of a new reference is most easily actioned by utilising Agenda's macro facility. Even this solution may prove untenable if the text-base is the result of many references and the developer may need to resort to adding an unindexed category of reference numbers that can be used to interrogate an external database.

It is in the area of maintaining an audit trail of which items generated which rules or new items that we have to make use of some of Agendas less common facilities, including the macro facility. Figure 16 illustrates one method of simulating relational linkages. In this view, the text of two rules generated from the highlighted item in Figure 17, have been added as items in the view. Figure 17 shows these two rule items being marked (F7) and then made to depend (ALT F3) on the highlighted item. This procedure makes the item that originally inspired the rule to be the rule's item pre-requisite. Agenda allows us to

manipulate these dependencies in a variety of ways through views. These manipulations are perhaps, best packaged as a macro to avoid mis-keying, and to speed up the display process.

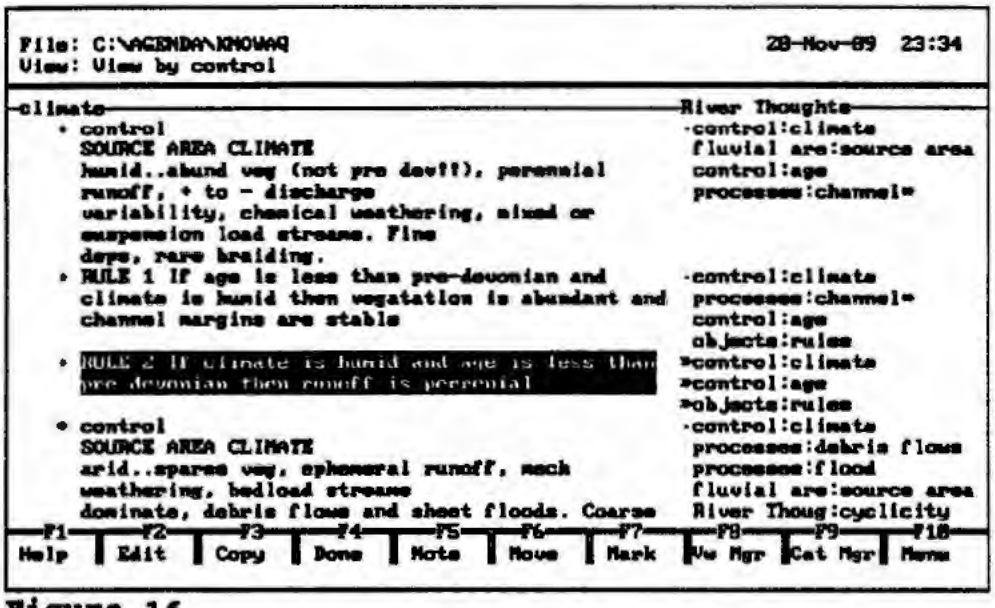
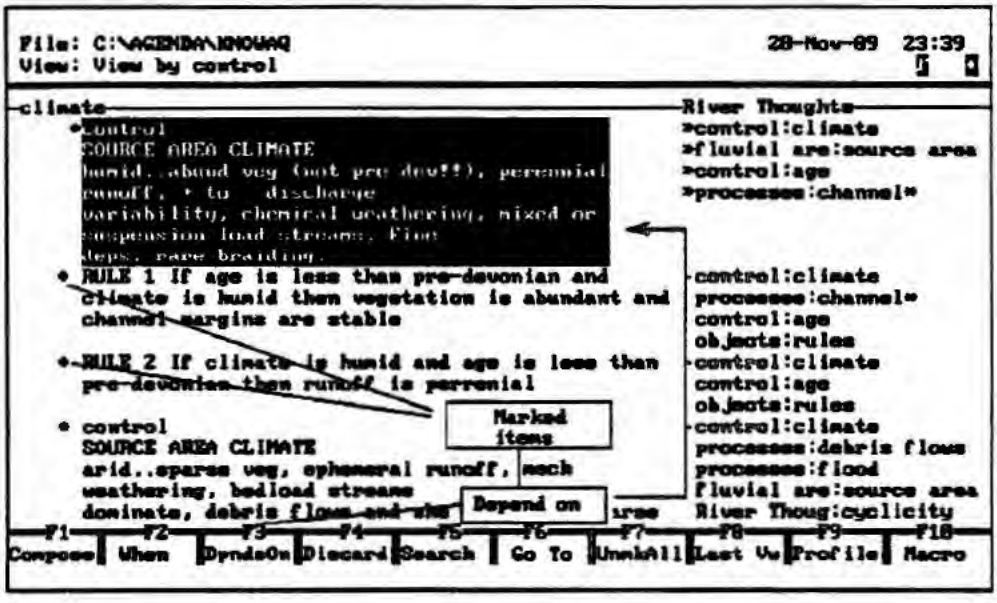


Figure 16
Two rules generated from highlighted item

Figure 18 displays the original highlighted item with an 'Attached rules' macro about to be invoked. This macro sets up a 'show view' via the utility menu that collects all the dependents of a given item or group of items. Figure 19 shows the macro pausing for user input having displayed the required view containing the two rules generated from the original item. Note, that the item has not been assigned to a rule category in this example although this would be possible. In fact it would provide yet another way of organising a view of items and rules, with both rule text and item text being grouped under unique rule section headings. Figure 20 shows a view of current rules in the text-base.



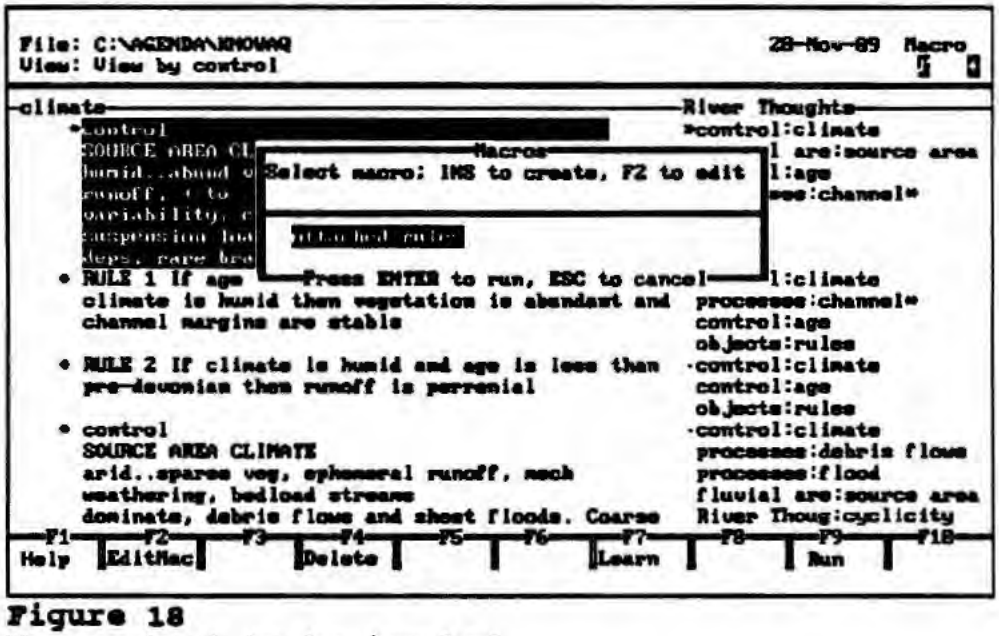


Figure 18
Macro about to be invoked

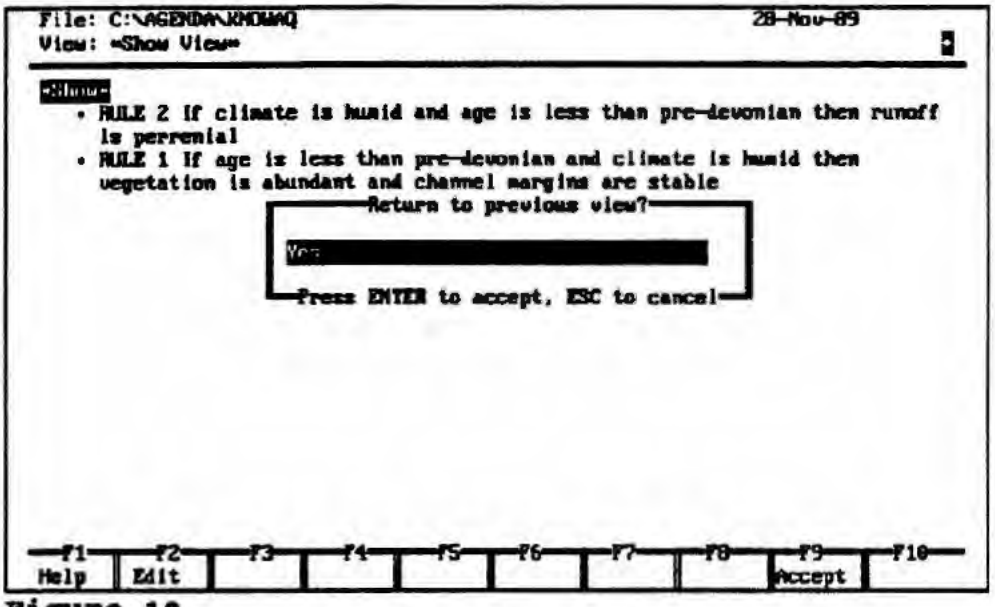


Figure 19
Macro pausing for instructions from user

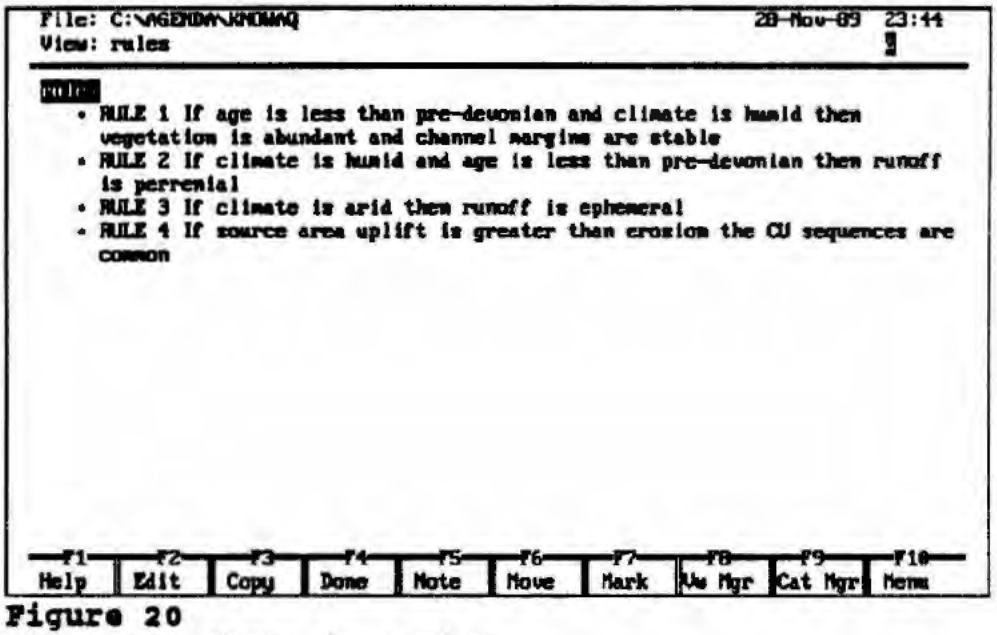


Figure 20
All rules shown in a view

6. ALTERNATIVES TO RAPID-PROTOTYPING

Breuka and Wielinga (1988) blame the use of ad hoc solutions to problems encountered in the early (ie knowledge acquisition) stages of expert system development, for the problems encountered in final delivery systems. They believe that the method of rapid-prototyping expert systems, advocated in the early 1980's (Hayes-

Roth et al 1983), is partly responsible for the lack of progress towards formal knowledge acquisition techniques. The solution they propose (1987) to expert knowledge elicitation is a formalised analysis of the domain and tasks to be performed that can provide generic interpretation models or templates as an intermediate knowledge representation that the knowledge engineer can use as more information is obtained. Their proposal acts on different levels of knowledge, initially in a bottom-up manner from a Domain level, up through to a more abstract Strategic level. Once this analysis is completed additional information gathering proceeds in a top-down, model driven manner, from the Strategic level to Domain level. Figure 21 summarises the different levels.

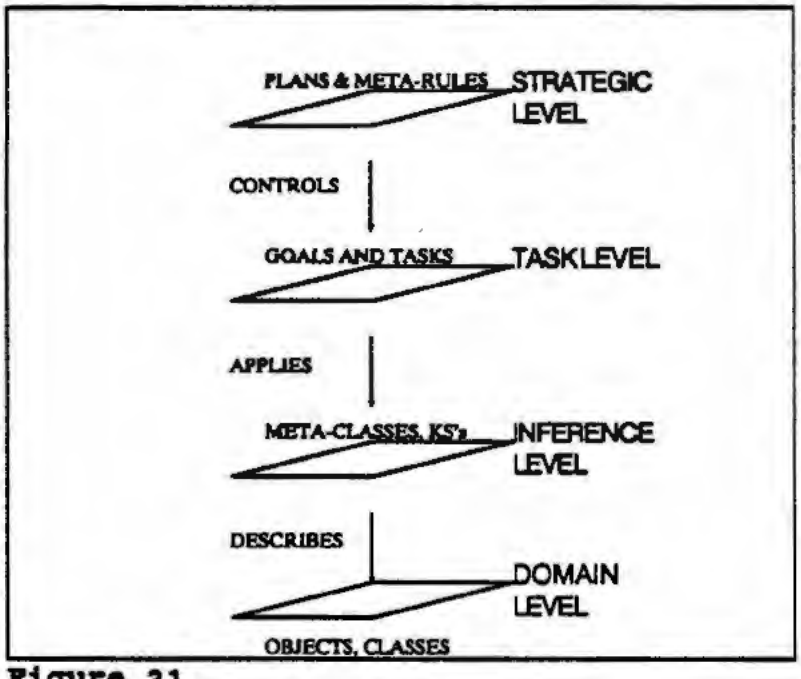


Figure 21
Knowledge levels

The Domain level identifies objects, concepts and relations that are largely task neutral. They have no part in solving a particular task, rather they characterise the domain in question at a fundamental level. The inference level identifies objects and relationships which, when applied to objects of the Domain level, result in directed reasoning to solve a domain task or tasks. The Task level actually defines those tasks that require action on the part of the system. The strategic level adds flexibility by providing a planning component to the system. In their 1988 paper they expand the basic concepts of their 'methodology' into generic interpretation models embodied in the KADS system, and define primitives for such models at an epistemological level. These primitives are objects, knowledge sources, models, structures and strategies and are based on work performed by Clancey (1983). Objects can be data or conclusions of inference processes. Classes of objects are data, hypotheses,

physical objects. A knowledge source is some set of knowledge that infers new objects from given objects. They are formalised in terms of rules and frame structures representing classification, specification, assembly and disassembly. Models may be related to support knowledge or have a function similar to a knowledge source. Such models may be causal, where processes are causally related; process models, where objects are related in a conditional manner; formal models, based on mathematical or physical theorems. Structures organise both objects and knowledge sources. Strategies are plans for invoking problem solving knowledge sources.

It is possible to implement some of these formal techniques in Agenda even as it is being used for rapid-prototyping. It is a simple task to include a "knowledge layer" category family, breaking down into domain, inference, task and strategy levels and assign items that fulfill the various criteria. By using the audit-trail technique described for rule generation from items in section 5.1.1 relationships between generated and generating items could be tracked and documented. The benefits of a side-by-side formal and informal approach are that the implications of each method can be viewed separately or together under the common control afforded by Agenda. Also, the effects of reclassification, addition of data, generation of new items is immediately available in the system for inspection and assessment. In fact, aspects of the knowledge levels, or Breuka and Wielinga's later levels of knowledge interpretation have already been mentioned in section 5.1.1.

Using a non-naive knowledge engineer can largely by-pass the need for their knowledge identification stage as the fundamental lexicon of domain concepts is already apparent to the developer. Agenda allows the developer to enter items relating to all aspects of the system, from domain objects, such as individual lithofacies, to abstract concepts regarding environmental controls, thus providing an objective medium to manipulate information through all knowledge levels. Semantic information and task specific items may in turn be generated from the more abstract concepts, and this can be isolated in the category hierarchy.

The consistent treatment of data-items has the effect that printed information from Agenda is a very useful intermediate knowledge representation. The views can be filtered to exclude all task specific items allowing the developer to concentrate on more general models. Or the opposite view could be taken to enable an independent expert to assess, add to or amend existing domain specific items. Implementational categories can be added as tool-kit specific issues are explored. Views excluding this information can be used in conjunction with external experts without having to resort to discussing rather cryptic tool-kit code that could obscure the underlying principles. This allows the knowledge engineer to operate through all levels of the

system while being able to present arguments in a readily digestible form for informed comment.

7. Conclusions

It is, perhaps, unwise to consider working with Agenda as a series of well defined steps progressing neatly from data-capture through data-input and category formulation to data-analysis. This is because the environment provided by Agenda allows rapid and dynamic manipulation of our data in many ways. Also, we may ask where does the knowledge acquisition stage of expert system production end? It seems reasonable that in poorly controlled domains the specification of a system is liable to change even during the implementation stages as new evidential information is discovered and it is precisely in this area that the impact of new information on the existing data can be readily assimilated using Agenda.

The FLEAS system is based in a poorly understood domain and it is unlikely that such a domain could be quantified in even the simplest terms. Therefore, the system that will eventually emerge will be incomplete but will serve as an approximation of the domain. This first approximation will lead hopefully to a deeper understanding of the problem area by providing experts with a fresh theoretical perspective on the field. By using Agenda as a tool for knowledge acquisition information pertaining to the implementational aspects of the system, from data objects through to plans and models can be explored, and manipulated in a consistent way. The advantage of this is that any semantic element of items of the text-base is separated into the category hierarchy and the relationships between different knowledge levels are made explicit by using the views for data exploration. This enables experimentation with knowledge structures without first having to enter them into Nexpert Object.

Acknowledgements
The FLEAS project is partly supported by Compaq Computer UK Ltd.

Trademarks

Agenda	Lotus Development Corporation
Nexpert Object	Neuron Data
Grandview	Symantec Corporation
IZE	Persoft
PC-Outline	Brown Bag Software
Tornado	Micrologic
WEB	Octave Associates

References

Breuka, B.J., Wielinga, J.A. 1988 'Use of Models in the Interpretation of Verbal Data', Knowledge Acquisition for Expert Systems, Kidd (Ed.), Plenum Press 1988.

Clancey, W.J, 1983 'The Epistemology of Rule-based Expert Systems: A Framework for Explanation', Artificial Intelligence, Vol. 27:289-350.

Grashion, A.R. 1988 'Computer-Aided Analysis of the Braided River Depositional Environment', Computer Department Technical Report CDTR 88-1, Staffordshire Polytechnic, Stafford, England.

Hayes-Roth, F, Waterman, D.A., and Lenat, D.B. (editors) 1983, Building Expert Systems, Addison-Wesley 1983.

Kidd, A.L., Sharpe, 1988 'Goals for Expert System Research: An Analysis of Tasks and Domains.', Research and Development in Expert Systems IV, Cambridge University Press 1987:146-152.

Wielinga, J.A., Breuka, B.J. 1987 'Models of Expertise', Advances in Artificial Intelligence-II, Du Boulay, Hogg and Steels (Eds.), Elsevier 1987

Young, R. 1987 'Role of Intermediate Representations in Knowledge Elicitation.', Research and Development in Expert Systems IV, Cambridge University Press 1987:287-288.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Knowledge Acquisition for Spatial Knowledge-Based System

Kazuo Hiraki and Yuichiro Anzai
Department of Computer Science
Keio University
3-14-1, Hiyoshi
Yokohama 223 Japan
E-mail: hiraki@aa.cs.keio.ac.jp

ABSTRACT

The paper presents a method of acquiring spatial knowledge for the management of spatial knowledge-based systems. By a spatial knowledge-based system, we mean a knowledge-based system for spatial knowledge. Our acquisition algorithm makes induction by generalization on positive or negative instances of spatial patterns, and outputs a constraint logic program that is declaratively an induced concept of the spatial patterns, and procedurally a procedure for the maintenance of consistency and reasoning about spatial relations. Thus our algorithm transforms spatial patterns to symbolic declarative/procedural representations. The maintenance of spatial knowledge-based systems is made by **C-PIG**, a constraint logic programming system developed in our laboratory.

1 INTRODUCTION

Representation and acquisition of spatial knowledge are crucial issues not only in Artificial Intelligence, but also for the progress of computer science. For example, in the fields of machine vision, computer graphics and computer-aided design, geometric and topological relations are represented and computed in various ways. Especially in the engineering domain, an efficient knowledge-based system is needed for managing the knowledge of spatial objects and spatial relations.

The following three points should be considered, when developing a knowledge-based system for managing spatial knowledge:

1. *Qualitative expressions*:
   Generally, an object's shape and location are represented by using numerical parameters. But in terms of tractability for symbolic inference systems, it should be possible to access the knowledge base by using qualitative expressions like *"left_of(a,b)"*, *"on(x,y)"*.

2. *Vagueness of spatial relation*:
   If we use first-order logic for representing spatial relations, a predicate that represents a spatial relation and its actual instance do not correspond in a one-to-one fashion. For example, there are many interpretations of the predicate *left_of(block_a,block_b)* (See Figure 1 from [Winston75]). There should be a mechanism to bridge the gap between symbolic expressions and varieties of actual instances.

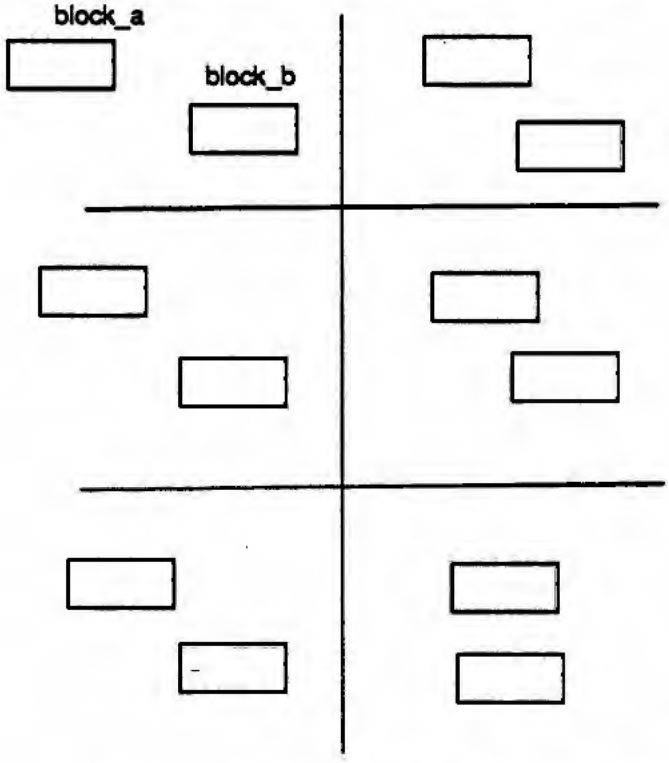


Figure 1 Vagueness of Spatial Relation [Winston75]

3. *Maintenance of consistency*:
   Updating the knowledge base while maintaining consistency is one of the most im-

portant tasks for the knowledge based system. For example, the system has to be able to compute the inconsistency between the assertions $left_of(a,b)$, $right_of(a,b)$. We need a mechanism that reasons about spatial relations in order to find such inconsistencies.

In addition to the above points, a knowledge acquisition method is needed to develop knowledge bases. When we cope with more complex relations, it is difficult to describe spatial knowledge *a priori*.

In this paper, we discuss the use of constraint logic programming language to develop a spatial knowledge-based system that satisfies above three points. We also describe a method of knowledge acquisition for the spatial knowledge based system. Section 2 describes a brief introduction to spatial reasoning by constraint logic programming and the architecture of the spatial knowledge-based system. Section 3 describes a method for acquiring spatial knowledge from positive or negative instances. By this method, the output spatial knowledge is represented as a constraint logic program. In Section 4, we explain how the acquired knowledge can be used in a spatial knowledge-based system. In Section 5, we compare our work with related research.

2 CONSTRAINT LOGIC PROGRAMMING AND SPATIAL KNOWLEDGE-BASED SYSTEM

2.1 Spatial Reasoning by Constraint Logic Programming

As mentioned in the above section, it is one of the most important tasks for a spatial knowledge-based system to manage knowledge bases while maintaining the consistency of spatial relations. For example, suppose that the following relations are entered into the knowledge base in this order:

```
left_of(a,b)
right_of(a,b)          (2-1)
```

A spatial knowledge-based system should be able to find an inconsistency between the two relations.

Next, suppose that the knowledge base has to answer the question "Is b above c ?" using the following knowledge (See Figure2):

```
left_of(b,a)
left_of(d,b)
below(c,a)
right_of(c,d)          (2-2)
```

A spatial knowledge-based should have a spatial reasoning mechanism in order to cope with these problems.

To compute the inconsistency in (2-1), we may possibly use the following explicit inference rule:

$$\forall X, Y \; left_of(X,Y) \wedge right_of(X,Y) \rightarrow inconsistent$$

However, when the number of spatial relations increases, the number of explicit inference rules becomes larger because of the combination of the relations. Suppose that there are N relations contradicting from each other, then $\frac{N(N-1)}{2}$ inference rules are needed for maintaining consistency of the knowledge base. Moreover, in situations like (2-2), inference rules become more complex because we have to deal a larger number of combinations of the spatial relation. Thus pre-defining such rules for reasoning about spatial relations is not easy at the present status of technology.

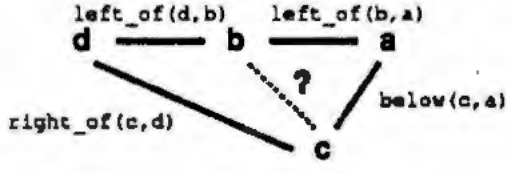


Figure 2 Example of spatial reasoning

Instead of using explicit inference rules, we can use constraint programming for spatial reasoning and finding inconsistencies. Constraint logic programming language (CLPL) can express and solve constraints. The difference between logic programming languages (e.g. PROLOG) and CLPL is that CLPL contains a module called the **constraint solver** that solves given constraints. Running systems have been constructed and applied to such areas as option trading[Lassez87] and scheduling [Dincbas88].

One of the most remarkable features of constraint-based language is that the user does

not have to write a procedure explicitly in order to solve constraints. The user has only to give *declarative* relations among variables.

This feature is useful for the management of spatial knowledge. We can use the **constraint solver** for maintaining consistency and reasoning about spatial relations. In the case of (2-1) for example, suppose that following constraint programs are given:

```
right_of( loc(A,[X1,Y1]), loc(B, [X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        Y =< -0.24*X,
        0.25*X =<  Y.                                    (2-3)

left_of( loc(A,[X1,Y1]), loc(B, [X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        -0.31*X =< Y,
        Y =< 0.24*X.                                     (2-4)
```

where the variables $X1$, $X2$, $Y1$, and $Y2$ represent the object location. We can find inconsistency between two assertions by calling the following CLPL goals:

```
:-   left_of( loc(a, LocA), loc(b, LocB) ),
     right_of( loc(a, LocA), loc(b, LocB) ).

       no                                                (2-5)
```

Note that the same variable $LocA$ ($LocB$) is used to represent the location of an object a (b). The programs (2-3) and (2-4) can be used not only for consistency maintenance, but also for spatial reasoning tasks like (2-2). Note that inequalities, $-0.31 * X =< Y$ in the program (2-4) for example, implicitly represent the *vagueness* of spatial relations.

2.2 Spatial Knowledge-Based System

The core part of the spatial knowledge-based system proposed in this paper was developed using a constraint logic programming language called **C-PIG** (Constraint Prolog Including Global variables). **C-PIG** is implemented as a meta-interpreter in Quintus

Prolog. **C-PIG**'s constraint solver can currently deal with linear-equations and linear-inequalities[1]. **C-PIG** is similar to CLP(R) [Lassez87] in regards to the ability of constraint solver[2]. However, **C-PIG** is equipped with the facility of global variable management, and thus can deal with global variables in addition to Prolog's conventional variables. This feature makes it easier to develop knowledge bases (See [Anzai89] for details).

Figure 3 shows the architecture of spatial knowledge-based system (called SKB hereafter) discussed in this paper. **Command interpreter** of SKB uses the following primitive commands:

1. assert_loc ··· Add a fact about global location to SKB.
2. assert_rel ··· Add information about relative relation to SKB.
3. delete_loc ··· Delete a fact about global location from SKB.
4. delete_rel ··· Delete information about relative relation from SKB.
5. ask_loc ··· Ask a question about global location.
6. ask_rel ··· Ask a question about relative relation.

SKB deals with commands $1 \sim 4$ while maintaining consistency of data, and answers commands $5 \sim 6$ by reasoning about spatial relations using **C-PIG**'s constraint solver. As described above, constraint logic programming can be a powerful framework for SKB. On the other hand, if we could acquire inductively constraint expressions from a scene that includes some spatial relations, we could develop knowledge bases about spatial knowledge automatically. **Learning Module** (called **Constraint Generator**) acquires spatial knowledge automatically by generating a constraint program of **C-PIG** from positive or negative instances of a spatial relation. Constraint expressions generated from Constraint Generator should be ones that can be computed by Constraint Solver included in **C-PIG**. We have assumed that a constraint that the Constraint Solver cannot handle must not be generated. Constraint Generator is implemented using Quintus Prolog. The learning algorithm of Constraint Generator is described in Section 3.

The first novelty of SKB is that the user supplies only instances of spatial relations to Learning Module instead of programs. The second novelty of SKB is its ability to reason about spatial relations by using constraints between variables even if there is no information about global locations. We can consider many kinds of applications by these advantages of SKB.

[1] We use *sup-inf method* to solve linear-inequalities [Shostak77].
[2] CLP(R) can also solve linear equations and linear inequalities.

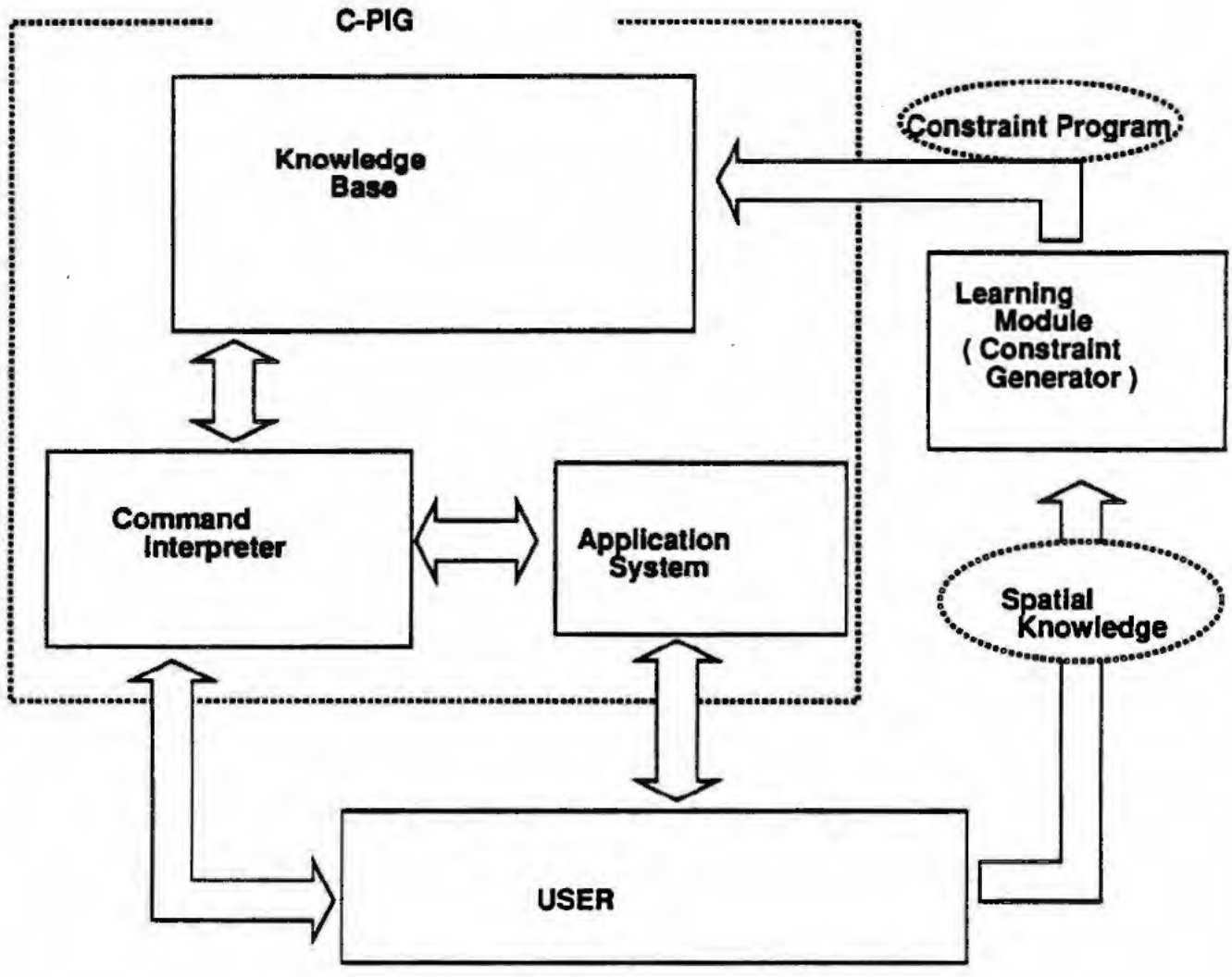


Figure 3 The Architecture of a Knowledge-Based System for Spatial Knowledge; SKB

3 A METHOD FOR ACQUIRING SPATIAL RELATIONS

In the above section, we gave an outline of SKB and the spatial reasoning. In this section, we show the detail of the method that generate a constraint program from positive or negarive instances of a spatial relation. The method can be divided into two parts. The first one uses the procedure (called generalization to interval) to generate an algebraic expression. The second one refined this expression by interaction with the teacher.

3.1 Generalization to Interval

The knowledge acquisition system described in this paper is an inductive system that generates an algebraic constraint expression of spatial relations from a finite sequence of positive or negative instances. In this section we describe our method of acquiring constraint expressions from positive or negative instances.

The system can deal with instances that are two points on E^2. Constraint Generator generates algebraic constraints on two parameters; the distance γ between the points and the direction θ from one point to the other. [3]

Input:

- Positive instances of a spatial relation R as ordered pairs of the points:
  $\{(P_1, P_2), (P_3, P_4), \cdots, (P_{i-1}, P_i), \cdots, (P_{2m-1}, P_{2m})\}$ $\quad P_i \in E^2$
- Negative instances of a spatial relation R as ordered pairs of points:
  $\{(N_1, N_2), (N_3, N_4), \cdots, (N_{j-1}, N_j), \cdots, (N_{2n-1}, N_{2n})\}$ $\quad N_j \in E^2$
- Relation name.

Assuming that the number of generated constraints on distance (direction) is p (q):

[3]For simplicity, we describe only the case of *binary-relations* in this paper. By combining binary relations, however, the argument in this paper can be extended easily to the case of *n-ary relations* that can be constructed from binary relations.

Output:

$$C_1 \vee C_2 \vee \cdots \vee C_k \vee \cdots \vee C_n$$

$$C_k \in \{$$

$$\begin{array}{ccccc}
(C_{\gamma 1} \wedge C_{\theta 1}), & \cdots, & \cdots, & \cdots, & \cdots, (C_{\gamma 1} \wedge C_{\theta q}), \\
(C_{\gamma 2} \wedge C_{\theta 1}), & \cdots, & \cdots, & \cdots, & \cdots, (C_{\gamma 2} \wedge C_{\theta q}), \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
\cdots & \cdots & (C_{\gamma i} \wedge C_{\theta j}), & \cdots, & \cdots, \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
(C_{\gamma p} \wedge C_{\theta 1}), & \cdots, & \cdots, & \cdots, & \cdots, (C_{\gamma p} \wedge C_{\theta q})
\end{array}$$

$$\}$$

where $C_{\gamma i}(C_{\theta j})$ is an algebraic description of distance (direction) and is called *atomic constraint*. *Atomic constraint* forms an interval of $\gamma\ (\theta)$:
$value1 \leq \gamma\ (\theta) \leq value2$. (If $value1 = value2$ then $\gamma\ (\theta) = value1$)

For example, suppose that the following constraint of a spatial relation, R_1, is generated:
$\{((1 \leq \gamma \leq 3) \wedge (\frac{1}{4}\pi \leq \theta \leq \frac{1}{2}\pi)) \vee ((1 \leq \gamma \leq 3) \wedge (\frac{5}{4}\pi \leq \theta \leq \frac{3}{2}\pi))\}$
then
$((0,0),(4,5)), \quad ((0,0),(-1,-5))$
are positive instances of R_1. Also
$((0,0),(4,3)), \quad ((0,0),(-2,-1))$
are negative instances of R_1.

The learning module generates disjunctions of conjunctive atomic constraints according to the following algorithm.

Algorithm1:

1. Generate:
$DP = \{dp_1, \cdots, dp_i, \cdots, dp_m\}$, $GP = \{gp_1, \cdots, gp_i, \cdots, gp_m\}$ from each ordered pair of positive instances where $dp_i = \|P_{2i} - P_{2i-1}\|$, $gp_j = arctan((y_{2i} - y_{2i-1})/(x_{2i} - x_{2i-1}))$, $P_{2j} = (x_{2j}, y_{2j})$, $DP, GP \neq \emptyset$.
And generate:
$DN = \{dn_1, \cdots, dn_j, \cdots, dn_m\}$, $GN = \{gn_1, \cdots, gn_j, \cdots, gn_m\}$ from each ordered pair of negative instances where $dn_j = \|N_{2j} - N_{2j-1}\|$, $gn_j = arctan((y_{2j} - y_{2j-1})/(x_{2j} - x_{2j-1}))$, $N_{2j} = (x_{2j}, y_{2j})$.
2. Generate an algebraic constraint expression by applying the procedure below called *generalization to interval* to each of DP, GP, DN, and GN:

Generalization to Interval

- For DP and DN :

  0: $H := \emptyset, p_1 := 0, I := sort(DP \cup DN)$

  1: $p := pop(I)$, If $I = null$ then return H

  2: If $p \in DP$ then $p_1 := p$ goto 1:
  else if $p_1 \neq$ then $push(0 \leq \gamma \leq p_1)$ to H

  3: If $I = null$ return H

  4: $p := pop(I)$

  5: If $p \in DP$ then $p_1 := p$, else goto 3:

  6: If $I = null$, $push(p_1 \leq \gamma \leq \infty)$ to H, return H

  7: $p := pop(I)$ If $p \in GP$ then $p_2 = p$, goto 6:,
  else $push(p_1 \leq \gamma \leq p_2)$ to H, goto 3:

- For GP and GN :

  0: $H := \emptyset, p_0 := 0, I := sort(DP \cup DN)$

  1: $p := pop(I)$, If $I = null$ then return H

  2: If $p \in GP$ then $p_0 := p$, goto 1:

  3: If $I = null$ return H

  4: $p := pop(I)$

  5: If $p \in GP$ then $p_1 := p$ goto 6:,
  else goto 3:

  6: If $I = null$ then

  if $p_0 \neq 0$ then $push(p_1 \leq \theta \leq p_0)$ to H
  else $push(p_1 = \theta)$ to H

  7: $p = pop(I)$

  If $p \in GP$ then $p_2 := p$ goto 6:
  else if $p_2 = null$ then $push(p_1 = \theta)$ to H, goto 3:
  else $push(p_1 \leq \theta \leq p_2)$ to H goto 3:

Figure 4 illustrates how distance values of positive or negative instances are generalized to intervals by the above algorithm. Values of positive instances (designated as p) 5, 15, 20 and values of negative instances (designated as n) 10, 25 are generalized to interval $0 \leq \gamma \leq 5$ and $15 \leq \gamma \leq 20$.

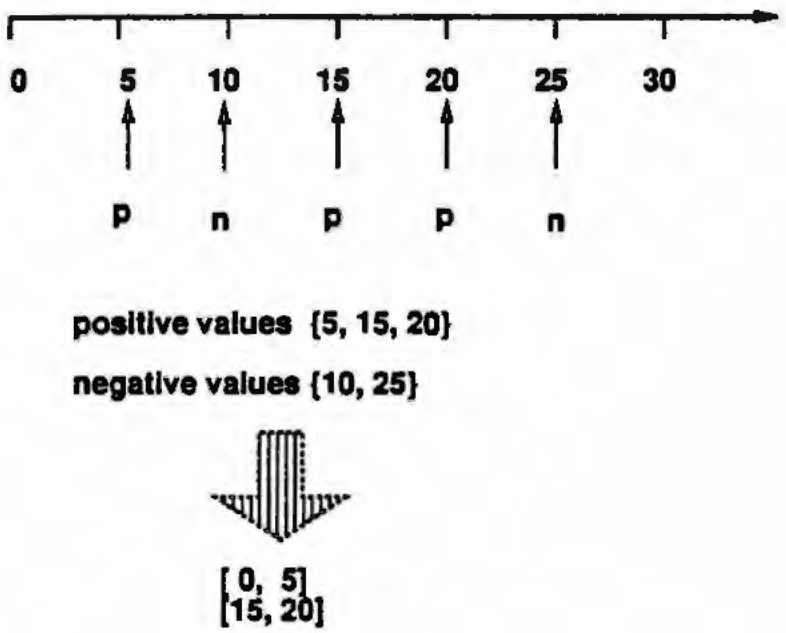


Figure 4 generalization to interval

For example, given positive instances of the spatial relation:
$\{((350, 253), (500, 207)), ((350, 258), (215, 211)),$
$((300, 295), (185, 367)), ((347, 240), (474, 324))\}$
and negative instances :
$\{((336, 257), (334, 197)), ((341, 238), (356, 392))\}$,
We get algebraic constraints on distance and on direction whose region is illustrated in Figure 5.

$$\begin{array}{rcl} ((155.724 \leq \gamma \leq \infty) & \wedge & (5.97635 \leq \theta \leq 0.580755)) \vee \\ ((155.724 \leq \gamma \leq \infty) & \wedge & (2.57599 \leq \theta \leq 3.45111)) \vee \\ ((136.213 \leq \gamma \leq 153.101) & \wedge & (5.97635 \leq \theta \leq 0.580755)) \vee \\ ((136.213 \leq \gamma \leq 153.101) & \wedge & (2.57599 \leq \theta \leq 3.45111)) \end{array}$$

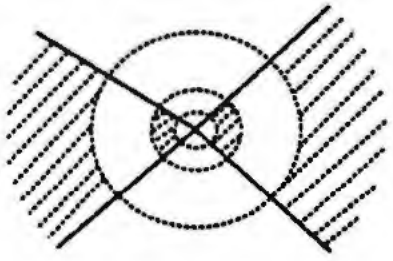


Figure 5 Region of constraint expression for the example problem

3.2 Modifying Constraints

From *Algorithm1* we can get constraint expressions by which all the previous positive instances, but none of the previous negative instances, are satisfied. However, desired constraint expressions for *next_to* should not have a constraint on distance. The previous example *next_to* depicted in Figure 5 shows the region of constraints as separated into concentric circles.

It should be noted that, since instances have two attributes, *distance* and *direction*, we cannot distinguish which attribute has a negative factor, *distance, direction, or both.*

For the above problem, we can get the desired constraint expression by asking whether the *inquiring instances* are positive or not to the teacher (who initially input these instances). *Inquiring* proceeds according to the following rules:

- *Select d_i from DN and any u that satisfy a constraint $C_{\theta i}$ generated from Algorithm1.*
  Inquire the teacher about whether the inquiring point (d_i, u) is positive or not.

If positive, then remove d_i from DN.

- *Select g_i from GN and any v that satisfy a constraint C_{γ_i} generated from Algorithm1.*
  Inquire the teacher about whether the inquiring point (v, g_i) is positive or not.
  If positive, then remove g_i from GN.

As for the previous example *next_to*, we get the following constraint expression by applying the above inquiring rules:

$$((0 \leq \gamma \leq \infty) \ \wedge \ (5.97635 \leq \theta \leq 0.580755)) \ \vee$$
$$((0 \leq \gamma \leq \infty) \ \wedge \ (2.57599 \leq \theta \leq 3.45111))$$

After acquiring spatial knowledge, the constraint description is transformed to C-PIG's program. C-PIG's constraint solver cannot deal with nonlinear equations and nonlinear inequalities currently[4] . Because acquired descriptions are nonlinear, we have to approximate these descriptions to linear equations or linear inequalities. The above constraint description of "*next_to*" is transformed to the following program:

```
next_to( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        Y =< -0.345895*X,
        0.361615*X =< Y.

next_to( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        -0.515502*X =< Y,
        Y =< 0.255538*X.
```

[4]C-PIG handles nonlinear constraints by using a delay mechanism until a sufficient number of variables are instantiated and the constraints become linear.

where "loc(A,[X1,Y1])" represents the location of the object A in E^2, and "Y =< $-0.345895 * X$" represents the inequality $Y \leq -0.345895 * X$. Note that two clauses correspond to two conjunctive atomic constraints.

4 EXAMPLES

In this section we explain how acquired constraint programs are used in the spatial knowledge-based system (SKB). Especially, we show how to maintain the consistency of data and to reason about spatial relation.

Suppose that we have the following C-PIG's program acquired by **Learning Module** [5]:

```
next_to( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        Y =< -0.34*X,
        0.36*X =< Y.

next_to( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        -0.51*X =< Y,
        Y =< 0.25*X.

below( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        -2.5*X =< Y,
        1.6*X =< Y.

right_of( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        Y =<  -0.24*X,
        0.25*X =< Y.
```

[5]Note that the normal variable scope in C-PIG's program is treated in the same way as in Prolog: the same capital letter used in different clauses are different variables in C-PIG.

```
left_of( loc(A,[X1,Y1]), loc(B,[X2,Y2]) ):-
        X = X2 - X1,
        Y = Y2 - Y1,
        -0.31*X =< Y,
        Y =< 0.24*X.
```

Also assume that the following relations have already been added to SKB:

```
assert_rel(next_to,a,b).
assert_rel(below,c,a).
assert_rel(left_of,d,b).
```

Note that there is no information about global location. Then, suppose that the following command is input to SKB:

```
ask_rel(right_of,c,b)
```

Command interpreter invokes constraints about objects *b* and *c* (next_to, below, left_of), and also the constraint currently questioned (right_of) as **C-PIG**'s goal inputs at the same time. Then, SKB examines whether these goals are satisfied or not. Computation for this question corresponds to solving the following goals:

```
:- next_to( loc(a, LocA), loc(b, LocB) ),
   below( loc(c, LocC), loc(a, LocA) ),
   left_of( loc(d, LocD), loc(b, LocB) ),
   right_of( loc(c, LocC), loc(b, LocB) ).   % constraints about b and c
```

SKB executes the command and returns *yes* as the answer. Note that the same variable names are used for representing the location of the same object. In this way, constraints are associated with input constraints about *b* and *c* by the unification mechanism.

In the same way, updating the database can be performed while maintaining consistency. For example:

```
assert_loc(right_of,d,c)
```

is found to fail by executing the following goals:

```
:- next_to( loc(a, LocA), loc(b, LocB) ),
   below( loc(c, LocC), loc(a, LocA) ),
   left_of( loc(d, LocD), loc(b, LocB) ),
   right_of( loc(d, LocD), loc(c, LocC) ).   % constraints about d and c
```

5 RELATED WORK

There are many previous studies that are concerned with spatial knowledge. Some of those studies typically use constraint-based programming for representing, and reasoning about, spatial knowledge. For example, in a study of image understanding, Waltz used constraint-like expressions to represent relations among vertices of blocks to find consistent labeling [Waltz75]. Brooks used constraints in a model-based vision system ACRONYM for modeling variations of structured objects [Brooks81]. As for spatial reasoning that can deal with qualitative expressions, Haar used fuzzy truth values for solving layout problems [Haar82]. Nishida [Nishida89] proposed a potential model for reconstructing spatial configurations from given natural language descriptions. These systems have particular mechanisms that solve constraints.

Comparing with these works, one of the novelties of our work is to use a constraint logic programming language **C-PIG** for representation and reasoning. Since **C-PIG** has a constraint solver that is independent from each problem, we do not have to huild a particular constraint satisfaction mechanism.

There are other types of constraint-based languages [Sussman80], [Leler88], [Borning81] that have a constraint solver as their internal mechanism. One of the advantages of using a constraint logic programming language is that semantics can be formulated more easily than other constraint-based languages, based on logic programming semantics[6].

On the other hand, the system described in this paper is an inductive system that generates algebraic constraint expressions of spatial relations from a finite sequence of positive or negative instances. There are some significant works in inductive learning. For example, Mitchell's Version Space [Mitchell82] is useful for finding maximally specific conjunctive generalization. However the method used in Version Space is restricted essentially to discrete domains. It has to be extended to continuous domains if we apply it to spatial

[6]See [Jaffer87] for further discussion.

relations.

As for systems for learning spatial knowledge, [Winston83] describes a synthesis of ANALOGY and ACRONYM for learning physical descriptions. [Connell85] deals with the generation of symbolic structure from given image data. These two systems have procedures for recognizing spatial relations before learning. On the contrary, our Learning Module (Constraint Generator) does not need such procedures and generates constraint descriptions of spatial relations.

6 CONCLUSION

In this paper, we have described a method of acquiring constraint expressions from positive or negative instances of spatial relations. We have also explained how acquired descriptions are used in a spatial knowledge-based system (SKB). SKB, combined with our knowledge aquisition mechanism, can be used for a lot of applications such as layout problems, computer graphics and modeling of physical systems [Gelsey89]. We are currently applying SKB to floor plannning problems.

We have concentrated in this paper on spatial relations that assume that a spatial object is a point. Actual objects in the real world, however, usually have structure. We are now also extending our idea presented here for spatial objects that have structure.

References

[Anzai89] Anzai,Y. and Hiraki,K.: Problem solving based on hypothesis generation and pattern information processing, In F.Mizoguchi, K.Furukawa and Y.Anzai (eds.), *Qualitative Reasoning*, Kyoritsu Shuppan Pub., 1989, pp.153-174.

[Brooks81] Brooks,R.A.: Symbolic reasoning among 3-D models and 2-D images, *Artificial Intelligence*, **17**, 285-348, 1981.

[Borning81] Borning,A.: The programming language aspects of ThingLab: A constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, **3**, 353-387, 1981.

[Connell85] Connell,J.H. and Brady,M.: Generating and generalizing models of visual objects, MIT A.I.Memo 823, MIT AILab, July, 1985.

[Dincbas88] Dincbas,M., Van Hentenryck,P., Simmonis,H., Aggoum,A., Graf,T. and Berthier,F.: The constraint logic programming language CHIP, *Proc. of the International Conference on Fifth Generation Computer Systems*, 693-702, 1988.

[Gelsey89] Gelsey,A.: Automated Physical Modeling, *Proc. IJCAI-89*, 1225-1234, 1989.

[Haar82] Haar,R.L.: Sketching: Estimating object position from relational descriptions, *Computer Graphics and Image Proccessing*, **19**, 227-247, 1982.

[Hiraki89] Hiraki,K. and Anzai,Y.: Context-dependent qualitative spatial expressions and their implementation to natural language dialogue systems, *Computer Software* (Journal of the Japan Society for Software Science and Technology), *6*, 56-64, 1989.

[Jaffer87] Jaffer,J. and Lassez,J.: From unification to constraints, In Furukawa,K., Tanaka,H. and Fujisaki,T. (eds.), *Logic Programming '87, Lecture Notes in Computer Science*, Springer, 1987.

[Lassez87] Lassez,C.: Constraint logic programming and option trading, IBM Technical Report, 1987.

[Leler88] Leler,W.: *Constraint Programming Languages: Their Specification and Generation*, Addison-Wesley, 1988.

[Mitchell82] Mitchell,T.: Generalization as Search, *Artificial Intelligence*, **18**, 203-226, 1982.

[Nishida89] Nishida,T., Yamada,A. and Doshita,S.: Figuring out most plausible interpretation from spatial constraints, In Kak,A. and Chen S. (eds.), *Spatial Reasoning and Multi-Sensor Fusion: Proc. of 1987 Workshop*, Morgan-Kaufmann, 158-167, 1987.

[Shostak77] Shostak,R.E.: On the SUP-INF method for proving Presburger formulas, JACM, **24**, 529-543, 1977.

[Sussman80] Sussman,G.J. and Steel,G.L.: CONSTRAINTS - A language for expressing almost-hierarchical descriptions, *Artificial Intelligence*, **14**, 1-39, 1980.

[Winston75] Winston,P.H.: Learning structural descriptions from examples, In Winston,P.H. (ed.), *The Psychology of Computer Vision*, McGraw-Hill, 1975.

[Winston83] Winston,P.H., Binford,T.O., Tatz,B. and Lowery,M.: Learning Physical Descriptions From Functional Definitions, Examples, and Precedents, *Proc. of AAAI-83*, 433-439, 1983.

[Waltz75] Waltz,D.: Understanding line drawings of scenes with shadows, In Winston,P.H. (ed.), *The Psychology of Computer Vision*, McGraw-Hill, 1975.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Models of Problem Solving in Knowledge Based Systems: Their Focus on Knowledge Acquisition

Wemer Karbach, Marc Linster and Angi Voß

Expert System Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1

e-mail: karbach@gmdzi.gmd.de, linster@gmdzi.gmd.de, avoss@gmdzi.gmd.de

We compare different approaches to the explicit modeling of problem solving processes in knowledge based systems to find out how they model knowledge, how they represent it, how flexible they are and to what degree they guide or support the development of knowledge based systems. We focus on Generic Tasks, KADS, role-limiting-methods and the method-to-task approach.
For each approach we describe key concepts, motivations, premises, claims and achievements. These are used to compare the terminology of the approaches, and to establish a mediating terminology for the rest of the paper. The prominent models are compared to show which problem areas they cover, and what methods they use. We analyze each of the approaches under the following hypotheses:

- it is useful to describe problem solving on a higher, more abstract level,
- models of problem solving can be specified in a problem specific, but application-task independent way,
- models guide the knowledge acquisition process and structure the knowledge base.

1. Introduction

The term model has widely been used in the construction of knowledge based systems. A model is a purposeful abstraction that allows us to reduce complexity by focusing on certain aspects. Researchers in AI have examined models to represent qualitative process relations [Forbus, 1984], qualitative dynamics [Kuipers, 1984], temporal relations [Allen, 1983, Voß and Linster, 1983], domains [Kodratoff and Tecuci, 1988, Morik, 1987] and problem solving [Breuker et al, 1987, Bylander and Chandrasekaran, 1987, Karbach, Voß and Tong, 1988, McDermott, 1988, Musen, 1988]. For our purpose we will focus on explicit models of problem solving and their role in the development of knowledge based systems.

All model based approaches to problem solving rely on the following hypotheses:

(1) It is useful to describe problem solving behavior on a more adequate, abstract level than the one offered by general purpose knowledge representation languages. These high-level descriptions are called models of problem solving;

(2) Models of problem solving behavior can be specified in a problem specific, but application task independent way;

(3) Models guide the knowledge acquisition process and structure the knowledge base.

To illustrate the general direction research is taking and, at the same time illustrate the fundamental and terminological differences between the approaches that make up the state of the art, we picked four advocates: Generic Tasks [Brown and Chandrasekaran, 1989, Bylander and Chandrasekaran, 1987, Chandrasekaran, 1986], KADS [Breuker et al, 1987, Breuker and

Wielinga, 1989, Schreiber et al, 1988], the method-to-task approach and the role-limiting-method approach. The label "method-to-task" was chosen for the approach taken in the PROTÉGÉ/p-OPAL combination [Musen, 1988, Musen, 1989], and we hope that "role-limiting-method" approach is a good characterization of the modeling technique underlying SALT, MOLE or KNACK [Eshelman, 1988, Klinker et al, 1989, Marcus, 1988, Marcus, 1989, McDermott, 1988]. We chose labels for the approaches rather than using names of systems, as we think that the different systems are only instantiations of the approaches and do not render their full potential. As most approaches have been developed by teams of researchers, names of authors would not do either.

Models of problem solving have sprung from two roots:

(1) Systematic knowledge engineering uses and develops problem solving models to analyze a problem at hand;

(2) Automatic knowledge acquisition is guided by the semantics of well-defined models of problem solving.

Our choice of approaches represents both roots: KADS focuses on systematic support of knowledge engineering in a software-engineering-like fashion and Generic Tasks makes several models available for reuse while the method-to-task and role-limiting-method approaches show that there is strong correlation between models and acquisition.

For our comparison we synthesized information from the different papers in the bibliography. In general we will not give any more explicit references. Assuming some familiarity with the work discussed, we will only give short surveys of the individual approaches: their key concepts, premises, claims and achievements. With this preparation, we can compare basic terms like *task* or *conceptual model* that recur with most authors. We will see that there are more or less significant deviations in the way they are used. We will establish our terminology for this paper by attempting a synthesis.

Moving on more solid terminological grounds, we will then present the concrete models developed by the four approaches and compare them with respect to the problems they tackle and the problem solving methods they use. We will describe which parts of the models are generic, what kind of knowledge must be added by whom, whether the models are operational, if they can be combined or if there is a modeling language that allows the development of new models. This comparison is done to reveal common features, expose differences and to discuss the three hypotheses stated above, thus hopefully allowing for a future synthesis.

2. Survey of Four Major Approaches

We give short surveys of the Generic Task, KADS, method-to-task and role-limiting-method approaches to the model based development of a knowledge based system. For reasons of simplicity we use the following abbreviations: GT (Generic Tasks), MT (method-to-task) and RLM (role-limiting-method). To prevent the implication of any judgement, we will present them in alphabetical order of the abbreviations. For each, we summarize the key concepts, motives, premises, claims and achievements. With *key concepts* we mean characteristic, central notions, the *premises* represent working hypotheses and underlying assumptions, and *claims* comprise advantages, purposes or goals of the approach. Under *achievements* we sum up what has been accomplished so far.

2.1. The Generic Task Approach

Papers underlying the presentation: [Bylander and Chandrasekaran, 1987, Chandrasekaran, 1986, Chandrasekaran, 1988, Brown and Chandrasekaran, 1989].

2.1.1. Key Concepts

Interaction problem: Knowledge cannot be represented free of assumptions of how it will be used, e.g. the form and the representation in which knowledge is needed for design is different from the one needed for diagnosis.

Generic task: A generic task is characterized by its input and output. It is a way to organize and represent the knowledge needed to solve certain kinds of problems, including a vocabulary of knowledge types and the process used as a problem solver.

2.1.2. Motives

Separating knowledge and inference, as it is done in universal languages, leads to systems which are at too low a level of abstraction, causing task level phenomena to be lost in details and raising implementation level phenomena to a theoretical status. Such systems are either inefficient or encode control aspects in the knowledge base.

2.1.3. Premises

The interaction hypothesis states that knowledge cannot be represented free of assumptions of how it will be used. This interaction problem can be exploited to select portions of domain knowledge and encode them into an efficient and maintainable problem solving structure, i.e. the generic task.

2.1.4. Claims

Building blocks of reasoning strategies can be identified that are generic and widely useful as components of complex reasoning tasks. With the identification of generic tasks languages can be developed that encode both the problem solving strategy and the knowledge that is appropriate for solving problems of that type. These languages facilitate the development by giving the knowledge engineer access to tools which work closer to the level of the problem, not on the level of implementation languages such as rules or frames.

2.1.5. Achievements

So far, several generic tasks have been identified and several languages were developed, in particular for the definition of diagnostic and routine design tasks.

2.2. The KADS Approach

Papers underlying the presentation: [Breuker et al, 1987, Breuker and Wielinga, 1989, Schreiber et al, 1988, Schreiber et al, 1987].

2.2.1. Key Concepts

Domain layer: The knowledge of the domain is described in terms of concepts and relations.

Inference layer: The inference layer specifies the basic inferences that are required to solve a problem using the domain knowledge. It uses domain-independent problem solving terms, e.g. *hypothesis*, *solution*, *abstract* or *refine*. It describes what inferences can be made, but not how or when they are made. It consists of meta-classes and knowledge sources which together constitute the *inference structure*. A *knowledge source* is a primitive inference making-function using knowledge provided by domain relations. Examples for knowledge sources are: *abstract*, *refine* and *match*. Its input and output parameters are *meta-classes*, which describe roles that domain concepts can play in a problem solving process, e.g. *solution* or *hypothesis*.

Task layer: It specifies the task decomposition used to control the inferences in terms of tasks and goals. A task combines knowledge sources and subtasks in order to achieve a particular goal. A fixed task structure realizes a particular strategy to solve a range of problems.

Strategic layer: On this layer the execution of the tasks is controlled and monitored and impasses of the problem solving process are diagnosed and repaired. This layer may be absent if the range of problems can be solved by using a same strategy (fixed task structure) and if the interaction between communication tasks and problem solving tasks is simple.

Conceptual model: This is a term borrowed from Norman [Norman, 1983] that denotes the model the designer of a computer system has in mind and/or specifies when implementing a system. In KADS it models the expertise by structuring the required knowledge, inference competence and use of knowledge. KADS has a semi-formal modelling language for describing conceptual models on four different layers corresponding to the different types of knowledge: domain, inference, task and strategic layer.

Interpretation model: This is a model of the problem solving process in a class of tasks over a type of domains. It provides an initial structure to the data and allows for a mainly top-down data analysis process. The interpretation model comprises strategic, task and inference layer. That means it is a conceptual model without domain layer. Elementary interpretation models that are used frequently are called *generic interpretation models.*

Design model: A design model describes the transformation of a conceptual model into an implementation on the basis of external and user requirements. It consists of three layers describing resp. the function, the behavior and the structure of the artifact.

2.2.2. Motives

Rapid prototyping approaches are inadequate for the systematic development of large knowledge based systems as they do not strictly separate the analysis from the design and implementation phases. Developing a knowledge based system is essentially a modelling activity. It is a mapping from an understanding of behavior in the real world to the description in form of an artifact: an engineering construct. The analysis of knowledge should be model driven as early as possible, as this makes the process more efficient and more proficient.

2.2.3. Premises

Knowledge and expertise can be analyzed before design and implementation starts. It is possible to sail between the Scylla of natural language and the Charibdis of a formal language, and express a model of expert problem solving at the epistemological level. It is useful to distinguish several generic types of knowledge (domain, inference, task and strategic), corresponding to different roles that knowledge can play in reasoning processes. These types of knowledge can be organized in layers, which have only limited interaction.

2.2.4. Claims

Control knowledge can be specified in a domain independent way. It is possible to identify constraints and top-down refinement models to guide the knowledge acquisition process in the framework of a comprehensive knowledge engineering methodology with a life cycle model, a modelling language, tools and techniques.

2.2.5. Achievements

The KADS group developed layered modelling languages for conceptual models and design models. In their approach, a conceptual model can be built from scratch or by selecting interpretation models, combining them and adding the domain layer. KADS has a library of interpretation models from which the knowledge engineer can choose.

2.3. The Method-to-Task Approach

Papers underlying the presentation: [Musen et al, 1987, Musen, 1988, Musen, 1989].

2.3.1. Key Concepts

Conceptual model: The conceptual model provides a set of terms and relations that define the semantics of the knowledge base elements that the program displays, and that ascribe meaning to the data entered by the user.

Representation-based conceptual model: TEIRESIAS-like tools allow the entry of knowledge in basic knowledge representation formalisms like frames, rules or clauses.

Method-based conceptual model: Knowledge is entered in the terminology of the underlying problem solver, i.e. as observations, fixes, requirements.

Task-based conceptual model: Knowledge is entered in application specific terms, e.g. as illness, drug dose, blood cell counts.

Content knowledge: Content knowledge is domain knowledge that is used by a problem solver in the way that is predetermined by the problem solving process.

2.3.2. Motivation

The development of knowledge based systems should be facilitated by separating the task of the knowledge engineer from the task of the domain expert, each doing what he is best at.

2.3.3. Premises

Problem solving knowledge can be modeled separately from the content knowledge and domain knowledge requirements (i.e. the needed content knowledge) can be described in terms of types that must be instantiated.

2.3.4. Claims

In domains that involve a number of similar tasks it is possible to create a systematic domain terminology and a general model of the building blocks of the tasks of the domain, i.e. a task-based conceptual model. These can be instantiated for the various tasks. Thus the difficult part of the knowledge acquisition process (defining the inventory and the problem solving terminology) needs not to be repeated for every knowledge base in the same domain.

2.3.5. Achievements

The MT knowledge acquisition methodology distinguishes tools that define method-based conceptual models, and tools that specialize them to task-based conceptual models. PROTÉGÉ and p-OPAL are representatives of this methodology. For example in the domain of cancer therapy PROTÉGÉ defines the terminology and the relations needed to describe treatment protocols for cancer therapies; p-OPAL instantiates this knowledge for specific therapies. Thus the knowledge acquisition process is separated in two phases with separate responsibilities for the knowledge engineer and the domain expert.

2.4. The Role-Limiting-Method Approach

Papers underlying the presentation: [Eshelman, 1988, Klinker et al, 1989, Marcus, 1988, Marcus, 1989, McDermott, 1988].

2.4.1. Key Concepts

Problem solving: Problem solving is the identification, selection and implementation of a sequence of actions that accomplish a task within a specific domain.

Problem solving method: A problem solving method provides a means of identifying, at each step, candidate actions. It provides one or more mechanisms for selecting among candidate actions and ensures that the selected action is implemented. In the RLM approach a problem solving method is close to what traditionally was called an inference engine.

Weak method: This is a term borrowed from Newell. A weak method is a problem solving method that is only weakly constrained by task features; each method is potentially applicable to a broad set of task types. Weak methods typically achieve this generality with simple control structures that require additional control knowledge to be supplied before the method can be applied. They cannot provide much help in determining what knowledge needs to be collected or how it should be encoded.

Role-limiting method: A role limiting method can be viewed as a specialization of a weak method. It guides knowledge collection and encoding and predefines the structure of the task-related control knowledge the method can use. A role-limiting method typically consists of a simple loop over a sequence of 5 to 10 steps. Some of the steps operate on significant bodies of task specific knowledge. Within a step there is no control, that is, it makes no difference in what order the actions are performed.The role-limiting method itself has little, if any, task specific knowledge (see section 2.5 for the definition of *task* within this paper. It does not correspond to the use of *task* by the authors of the RLM approach).

2.4.2. Motivation

The need for a taxonomy of tasks with a good description of the various problem solving methods expert systems use and to clearly define how these descriptions help with knowledge acquisition.

2.4.3. Premises

Many, but probably not all tasks can be performed with modest amounts of task specific control knowledge.

2.4.4. Claims

Role-limiting methods have a broad scope of applicability and they provide substantial help in specifying what knowledge needs to be acquired to perform a task and how this knowledge can be appropriately encoded.

2.4.5. Achievements

Discriminating features of problem solving methods used in expert systems have been identified. Tools have been built that tackle classes of tasks, each of the tools knowing what sort of knowledge is required by the underlying problem solver.

2.5. Comparison of some Basic Common Concepts

The concepts of *task*, *problem solving method*, *implementation level*, a "more abstract level", *control knowledge* and *conceptual model* are basic to all four approaches, though they may not always be explicitly mentioned or occur under different labels (see table 1). The use of *task* in the literature illustrates this confusion: In (McDermott, 1988) it denotes a problem to be solved, in KADS a task is a procedure to obtain a goal and may use one or several knowledge sources (Breuker et al, 1987, p. 169), in (Brown and Chandrasekaran, 1989, p. 9) a task is defined as "...an elementary generic combination of a problem, representation and inference strategy".

Table 1: Comparison of the terms used by the authors of the different approaches. Key concepts are in italics.

Term	GT approach	KADS	MT approach	RLM approach
Task	Application of problem solving methods independently of a particular domain.	Colloquial use: application of problem solving methods to problems in a particular domain. Formal use: a task is a control construct on the *task layer* of the KADS *interpretation* and *conceptual models*.	Application of problem solving methods to problems in a particular domain. The reason to write an expert system.	Application of problem solving methods to problems in a particular domain.
Problem solving method	Here the terms "inference strategy" and "*(generic) task*"are used. The latter determines the knowledge representation (cf. *interaction problem*)	*Interpretation models* are descriptions of problem solving methods.	A domain independent description of how to reach a solution.	A domain independent description of how to reach a solution that it is expressed as a sequence of "identify, select and implement" steps.
Modeling level(s)	On the task level the generic tasks are formulated as a problem solving method with a knowledge representation.	Brachman's epistemological level [Brachman, 1979] is the level of development for the conceptual models. The models themselves consist of descriptions on four levels: domain, inference, task and strategy.	Two levels are distinguished: (1) development of a *task model* by the knowledge engineer at the PROTÉGÉ level; (2) instantiation of the *task model* by the expert at the p-OPAL level.	————
Implementation level	Rule/frame/logic level	The implementation level is the level where the knowledge based systems are programmed. This can be Prolog, a shell or any other language.	————	————
Control knowledge	————	On the *task layer* control is specified in a domain independent way.	Control and control knowledge is formulated in domain terms.	"... a method's control knowledge comprises (1) an algorithm specifying when to use what knowledge to identify, select and implement actions together with (2) whatever knowledge the method has for selecting among candidate actions."
Conceptual model	————	Colloquial use: the model of expertise that the knowledge engineer has in mind. Formal use: description of knowledge on four layers that will be implemented via the *design model*.	The *conceptual model* provides a set of terms and relations that define the semantics of the knowledge base elements that the program displays, and that ascribe meaning to the data entered by the user.	————

To establish the terminology of this paper, we attempt the following synthesis.

Task: We prefer the meaning chosen in the MT and RLM approaches, which also coincides with the colloquial use in KADS, i.e. a task is the application of a problem solving method to a problem in a particular domain. Examples are the application of skeletal-plan refinement to the development of individual treatment protocols in the domain of cancer therapy, or the application of hierarchical classification to fault-finding in four-stroke four cylinder engines of a particular manufacturer.

Problem solving method: Our intuitive understanding of the term coincides with the definition used in the MT approach, as it is more general than the one used in the RLM, GT or KADS approach. Generic tasks and interpretation models are special ways to describe problem solving methods.When trying to characterize problem solving methods by their control and control knowledge we saw the need to be more precise, as control is too fuzzy a notion. Extending the RLM definition we adopted the following meaning:

A *problem solving method*, or just method, is characterized by its types of actions, the selection between the types, the selection between the instances of the selected type and the execution of the selected action. The basic procedure consists of: (1) recognizing what type of action comes next; (2) which particular actions are available of that kind; (3) select one of those and (4) execute it.

Actions are the elementary steps executed by the method. An action may be completely task-neutral, or completely task-specific, or it may have some generic and some task-specific aspects.

Problem solving methods operate on data structures. The *data structures* manipulated by the actions are usually defined by the problem solving method, but sometimes they may be refined in a task-specific way, while the concrete *data* to be filled in are usually completely task-specific. If the next action to be executed cannot be determined in a task-independent way, task-dependent *selection criteria* must be provided.

We use the term *knowledge needs* to comprise both the action types and the data structures needed by a problem solving method.

Modeling level: The modeling level provides a problem solving specific language which may, but need not be operational. It describes and differentiates the roles of knowledge by abstracting from the underlying implementation formalism.

Implementation level: This is the level of general purpose languages that are not specialized to any problem solving methods. The borderline between modelling and implementation levels may not always be easy to draw.

Conceptual model: This is a model of a system builder about how knowledge is used by a person or a computer program and which roles it plays in that process. That means, conceptual models, or just models, may be mental as well as verbalized, and they may model the expert as well as the system or a completely new approach invented by the knowledge engineer.

Models of problem solving may consist of concrete methods and the knowledge they operate on, but they may also provide a more abstract view of the intended behavior of the system. In each case the model ensures communicability and transparency, whereas, if given, the method ensures operationallity.

3. Comparison of the Models

Each of the four approaches has elaborated several substantial problem solving models. We will give a short characterization of the most well known ones before we cluster them according to the problem class they tackle.

3.1. Description of the Major Models

We present the most prominent problem solving models of the four approaches according to a common scheme: *Problem class* specifies the major class of problems which can be tackled. We only distinguish analysis and synthesis. See section 3.2 for a definition. We do not use a finer distinction because we do not want to address the whole issue of problem type classification in this paper. The original classification by the respective authors is included in parentheses. *Method* gives the name of the problem solving method used and *description* a short verbal description thereof. *Knowledge needs* describe the information needed to supplement the problem solving method for a particular task. They specify the view on the world required to apply the models.

3.1.1. Generic Tasks

CSRL:

Problem class:	analysis (hierarchical classification)
Method:	establish-refine
Description:	If a node in the hierarchy is successful consider its children. If a node fails, reject it and prune the subtree.
Knowledge needs:	classification hierarchy, matching from features to concepts

DSPL:

Problem class:	synthesis (routine design)
Method:	plan refinement, skeletal planning
Description:	A predefined design plan is top-down refined. Knowledge for backtracking is available.
Knowledge needs:	hierarchical structure of the object to be designed, design plans for each node in the hierarchy, backtracking knowledge

HYPER:

Problem class:	analysis (hypothesis-matching)
Method:	matching
Description:	In a hierarchy of abstractions evidences are propagated bottom-up.
Knowledge needs:	hierarchy of feature abstractions, procedures to determine the evidence of high level abstractions

IDABLE:

Problem class:	analysis (knowledge-directed-information-passing)
Method:	abstraction
Description:	Knowledge is generated using inference procedures or default knowledge.
Knowledge needs:	data concepts with default values, methods for inferring data

PEIRCE:

Problem class:	synthesis (abductive hypothesis assembly)
Method:	cover-and-differentiate
Description:	Searches for a set of hypotheses which best explain a given situation.
Knowledge needs:	hypotheses, causal relationships between hypotheses, plausibility values

3.1.2. KADS' Interpretation Models

Systematic diagnosis: localization, causal tracing

Problem class:	analysis (diagnosis)
Method:	A system model is decomposed (component-wise resp. causally). Complaints are compared with standard values in each decomposition to identify faulty components resp. faulty states.
Knowledge needs:	system model, variable value, norm, difference, conclusion

Heuristic classification

Problem class:	analysis (diagnosis)

Method: The method is based on the "horse-shoe model" of Clancey [Clancey, 1985]. Observables are abstracted to reach a solution class via matching. This solution class is further refined into a solution.

Knowledge needs: observables, variables, solution abstractions, solutions

Assessment

Problem class: analysis (assessment)

Method: A case description is compared to a system model. Via abstraction and matching the discrepancy between both is inferred and classified into a decision class.

Knowledge needs: case description, abstract case description, norm, discrepancy, decision classes

Monitoring

Problem class: analysis (monitoring)

Method: From a system model a parameter set is selected to be monitored. Observables are compared to the parameters and in the case of discrepancies classified into a discrepancy class.

Knowledge needs: observables, variable values, system model, parameter values, differences, discrepancy classes

Prediction

Problem class: analysis (prediction)

Method/Description: From a description of the processes and objects of a system and the knowledge about their behavior the future state of the system is predicted (e.g. by default reasoning, simulation or constraint propagation).

Knowledge needs: augmented system description (= system description + system model), predicted system description

Prediction of behavior

Problem class: analysis (prediction)

Method/Description: The state of the system is predicted based on a causal system model.

Knowledge needs: system model, state description, augmented state description, parameter values, predicted state description

Design based on given abstract models (a general description)

Problem class: synthesis (design)

Method/Description: From informally specified requirements a formal description is inferred. An abstract model of the object to be designed guides the construction of the artifact.

Knowledge needs: informal problem statement, formal specification, conceptual model of the final product, detailed design

Hierarchical design

Problem class: synthesis (design)

Method/Description: The abstract model is decomposed into components. Each component is designed separately. The detailed component designs are then aggregated.

Knowledge needs: formal specification, skeletal model, components, model of component interaction, detailed model

Incremental design (only inference structure given)

Problem class: synthesis (design)

Method/Description: From a full specification given design elements can be chosen by transformation rules. They are composed using a skeletal model.

Knowledge needs: conceptual model of the detailed design, functions design elements, model class, skeletal model, detailed design

3.1.3. Method-to-Task Approach

PROTÉGÉ:

Problem class: synthesis (planning)
Method: skeletal-plan-refinement
Description: The method assumes the existence of a basic plan outline so that the system can concentrate on filling in the details.
Knowledge needs: Planning entities constitute a skeletal plan (classes, attributes, attribute properties).Task-level actions can modify the manner in which the planning entities are carried out. Input data is categorized in classes.

3.1.4. Role-Limiting-Method Approach

MOLE-p:

Problem class: analysis (diagnosis)
Method: cover-and-differentiate
Description: Candidate hypotheses are activated to explain complaints. Further observations are requested in order to differentiate between hypotheses.
Knowledge needs: states (complaints, hypotheses), causal relation between states

SALT-p:

Problem class: synthesis (construction)
Method: propose-and-revise
Description: Propose values of design parameters and check constraints on the values. If a constraint is violated apply fixes to revise the value by applying remedies.
Knowledge needs: value-dependency network, value constraints, fixes

SIZZLE-p:

Problem class:
Method: Case-based reasoning
Description: An appropriate case is chosen from a case library based on a similarity metric. There exists predefined knowledge on how to adapt a case to the new problem.
Knowledge needs: similarity between cases, extrapolation knowledge

YAKA-p:

Problem class: analysis (diagnosis)
Method: qualitative reasoning
Description: Abnormal values are identified with a model of normal system behavior. Traces over qualitative equations are used to identify a set of potential causes. This set is differentiated by a cover-and-differentiate method.
Knowledge needs: model of normal functioning system

3.2. Scope of the Models

We will now classify the models according to their scope, i.e. the problem class they are concerned with and the problem solving method they employ. To draw the distinction between analysis and synthesis, we look at the results of the implemented problem solving methods. If the method selects from a predefined set of solutions, then it is of analysis type. If it constructs its solutions, then it is of synthesis type. Obviously this distinction is a definitional one, may be it is artificial. MOLE can be viewed as a system that selects its solution, or as one that synthesizes an explanation path. Nevertheless it seems beneficial to consider the distinction between synthetic and analytic problem solvers.

All models for synthesis are based on a predefined structure of the artifact to be designed. *PROTÉGÉ*, *DSPL* and *Hierarchical design* use skeletal-plan refinement as problem solving method. *SALT* proposes design extensions using a data-dependency network. *Incremental design*

can also be seen to fall into this category. For problem solving methods in analytic problems, *CSRL* and *Heuristic classification* stand for an associative approach while *MOLE*, *PEIRCE* and *Systematic diagnosis* operate on causal networks. *Monitoring* and *Prediction* also fit into the analytic category.

SIZZLE, *YAKA* and *Prediction of behavior* are representatives of non-standard approaches to problem solving. Conventional problem solving methods tackle problems from scratch using shallow knowledge. *SIZZLE* allows the use of experience in problem solving by case based reasoning. *YAKA* and *Prediction of behavior* integrate model based reasoning based with deep knowledge. These three models can be seen as complementary to the problem solving methods in the previous paragraph.

Some of the proposed models, like *IDABLE*, *HYPER* and *Assessment* represent techniques widely used in AI. They are not related to a special task or problem solving method. For example, data abstraction -as in *IDABLE*- can be used in diagnosis to abstract data to features. In synthesis problems it can be used for abstracting features of a complete system on the basis of the features of each single component. These models can be seen as basic ingredients for problem solving methods. They are comparable to standard knowledge sources in KADS, although they can expand to a complete model as in *Assessment*.

4. Comparison of the Approaches

We will now examine the three hypotheses stated at the beginning of this paper.

4.1. Hypothesis I: *It is useful to describe problem solving behavior on a more adequate, abstract level than it is offered by general purpose knowledge representation languages*

The models sketched in the previous section specify problem solving methods on the modelling level. They have turned out to be useful descriptions because they are closer to the task and ease the communication between expert and knowledge engineer. Except for KADS, the models come with concrete problem solving methods directly corresponding to operational systems.

4.2. Hypothesis II: *Models of problem solving behavior can be specified in a problem specific, but application-task independent way*

To assess this hypothesis we will now analyze what knowledge in the model is specified in a task-independent resp. task-dependent way. When establishing our use of the term "problem solving method" in section 2, we took pains to identify potentially task-specific aspects of a problem solving method. In table 2 we separate the (kernel of the) *problem solving method* from its actions, data and *selection criteria*. We distinguish generic from instantiated actions and data structures from concrete data. *Generic actions* and *data structures* describe the format of *instantiated actions and concrete data*, respectively. We were unable to distinguish generic actions from the data structures and instantiated actions from the concrete data they operate on. This is partly due to poor descriptions in the literature and partly because in object-oriented languages the distinctions between data structures and action descriptions are muddled.

Table 2: How different types of knowledge are handled in the models of the four approaches.

Types of knowledge	GT approach	KADS	MT approach	RLM approach
Problem solving method	Several predefined languages with interpreters.	Interpretation models.	Predefined e-ONCOCIN interpreter.	Several predefined interpreters.
Generic Actions and the data structures they operate on	Each language provides a set of object-oriented definitions of generic actions and the data structures they operate on.	Defined as knowledge sources and meta-classes on the inference layer.	Planning entities are refined into task-specific generic actions and data structures in PROTÉGÉ.	For each interpreter a set of generic actions and data structures is predefined. They correspond to some of the roles of the method.
Instantiated actions and their concrete data	Domain specific knowledge filled into a predefined data structure is used by the interpreter to instantiate generic actions at run-time.	Defining a domain layer and linking it to an interpretation model yields a conceptual model.	Domain specific generic actions are instantiated in p-OPAL.	Domain specific actions can be defined according to the schema defined by the generic actions. At the same time data is filled in.
Selection criteria for generic actions	Fixed.	Domain independent, on the task layer.	Defined in PROTÉGÉ.	Fixed.
Selection criteria for instantiated actions	Domain specific knowledge.	None, all control happens on the task layer.	Defined in p-OPAL.	Domain specific knowledge.

In table 2, "predefined" entities belong to the generic part, while "defined" ones may be supplied by the knowledge engineer or the expert. In the MT approach, a very clear distinction is drawn between defining and instantiating. To elucidate it, we included the respective editors in the table.

To illustrate the labels in table 2 we give examples for each of the approaches:

GT For example in DSPL [Brown and Chandrasekaran, 1989] the generic actions are:

- Specialist action
- Plan action
- Task action
- Step Action
- Constraint action.

The instantiated actions correspond to the instantiated specialists, plans, tasks steps and constraints. For example, in the attribute *Design-Plans* of the specialist action "Head" corresponding plans are specified.

The distinction between selection criteria for generic actions and those used for instantiated actions are not so obvious here. Much of the generic selection criteria are encoded in small methods associated with the specialists and plans, e.g. "to select a plan, evaluate its sponsors and select the one with the highest rating".

KADS In the general framework of KADS it can be said that the generic actions are the knowledge sources, the generic data structures are the meta-classes. The selection criteria for generic actions are encoded on the task layer. If the knowledge sources are called in a fixed sequence the selection criteria can be compiled into a fixed task structure. In more complex cases that require dynamic decisions of the calls a hierarchical structure of goals can be used, each of those goals being obtained by the call of one or several knowledge sources.

The instantiated actions and data structures are the domain equivalences of the knowledge sources and meta-classes. Little can be said about this, as this mapping is hand-crafted during the design and implementation process.

RLM For example in SALT [Marcus, 1988, p. 83] three generic actions are identified:

- Propose a design extension
- Identify a constraint
- Propose a fix

These generic actions are called in a fixed sequence which represent the compiled selection criteria for generic actions. Each of these generic actions has a set of instantiated actions. We will illustrate this for "Propose a design extension".

Car-Jamb-Return is one of the design extensions known to the system. It is defined as:

Name:	Car Jamb Return
Precondition:	Door-Opening = Center
Procedure:	Calculation
Formula:	[Platform-Width - Opening-Width] / 2

Once the generic action "Propose a design extension" has been chosen by the generic control, the instance action car-jamb-return is applicable if the data-flow analysis allows it, i.e. if the values for the parameters of the formula are known and if the precondition is satisfied. These conditions are the selection criteria for the instantiated actions.

MT PROTÉGÉ/p-OPAL and the underlying reasoner ONCOCIN use the following generic actions:

- Check if a planning action is still valid
- Request tests
- Set parameter values
- Select successor planning actions.

If the generic action "request tests" has been selected by the generic criteria, then several tests can be selected, e.g. in the chemotherapy application the tests WBC (white blood cell count) or Platelet Count can be considered.

The selection criteria for the instantiated actions are: Is it the correct day in the cycle, is the test compatible with the current chemotherapy? For WBC part of the selection criteria could look as follows:

IF	current-chemotherapy = VAM
	Day-in-cycle = 1
	Chemotherapy-Delayed = false
Then	Select WBC

Table 3 explicates for all approaches which types of knowledge are generic, which must be supplied by the knowledge engineer and which by the expert. In all the approaches the problem solving method is predefined in a problem specific but task-independent way. And the parts to be filled in are always task-specific. However, the borderline between the generic and the task-specific parts varies. Except for the MT approach, the generic parts always include the selection criteria, generic actions and data structures. Complementary, in all approaches the instantiated actions and the concrete data must be entered. These remarks apply to the KADS approach only if predefined interpretation models are selected. Alternatively, KADS allows to construct models from scratch using the four layer modelling language. In this case, the modelling language is the only generic part, and the problem solving method (the interpretation model) must be specified

completely by the knowledge engineer. This brings us to the more fundamental differences of the four approaches.

Table 3: Origin of the different types of knowledge in the four approaches.

Generic vs. task specific	GT approach	KADS	MT approach	RLM approach
Generic part	Problem solving method, generic actions and data structures, selection criteria for generic actions.	Problem solving method or a more abstract description thereof.	Problem solving method.	Problem solving method, generic actions and data structures, selection criteria for generic actions.
Task specific knowledge filled in by KE	Data and selection criteria to instantiate actions.	Instantiated actions and data.	Generic actions and data structures, selection criteria for generic actions.	Instantiated actions and data, selection criteria for instantiated actions.
Task specific knowledge filled in by experts			Instantiated actions, selection criteria for instantiated actions.	Instantiated actions and data, selection criteria for instantiated actions.

We already criticized that the models are not always as explicit and developed as desirable. In fact, only KADS provides a (semi-formal) modelling language for describing arbitrary models, and this language was actually used to specify all the interpretation models mentioned in the previous section. The other three approaches all provide a set of isolated, hand-coded models. Each is described either using a particular modelling vocabulary or in prose that is not reflected in the implementation.

While KADS is the only approach providing a general modelling language, it is also the only one whose models are not operational. These are two sides of the same coin. The generality of the language makes it not directly implementable. As shown in figure 1 the gap between the modelling language and the implementation must be bridged by design models.

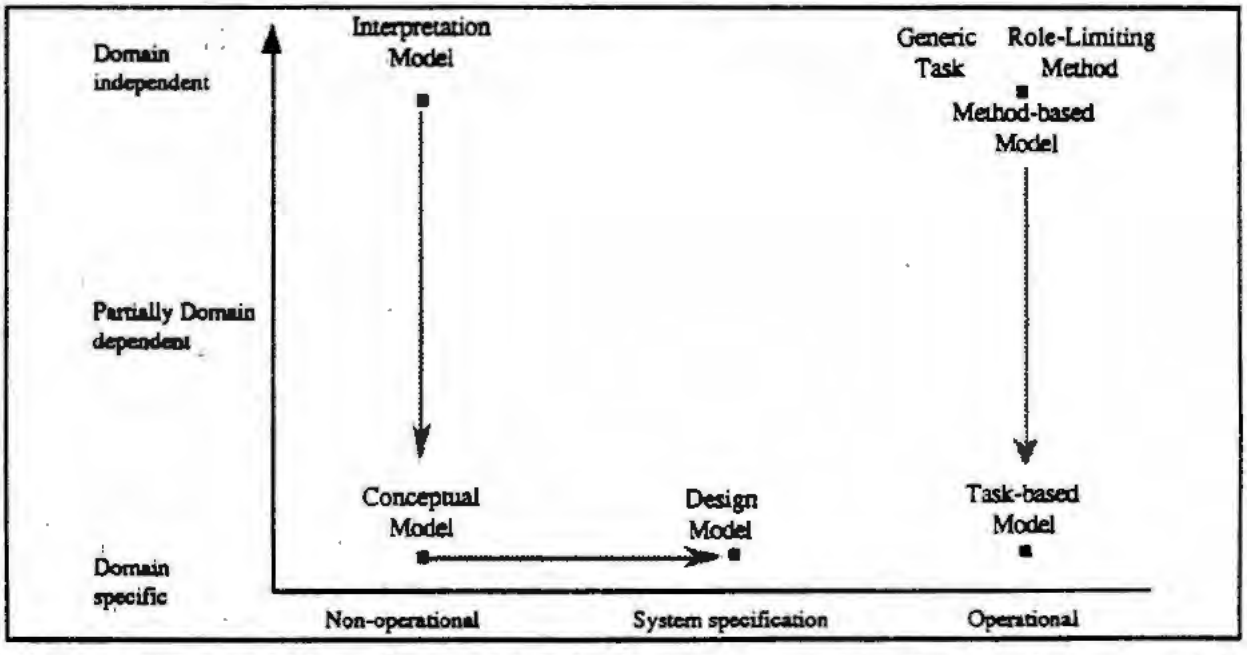


Figure 1: Different problem solving models ordered according to their level of specialization and their operationallity.

While the KADS conceptual model is supposed to model the expert rather than the system, the design model describes the system, though no longer on the modelling level. None of the other approaches directly models the expert, as they provide a fixed set of modelling level descriptions of operational systems. The problem solving model of the operational system may coincide with the expert's behavior.

If operational systems are written in the same implementation language they can be combined, even if a combination is not supported at the modelling level. E.g. generic tasks can communicate via message passing. This is different in KADS: Since there is a general modelling language and no implementation to be considered, their models are arbitrarily combinable.

Another crucial aspect of a modelling approach is its support for choosing the right model. All approaches describe the knowledge needs resp. input in prose. This corresponds to the task-specific knowledge in the latter two rows of table 3. Only the GT approach also specifies the output, and KADS offers the inference structures, i.e. a control-independent description of the elementary actions and their data structures. This seems to be a rather sad affair. A taxonomy of the various models with decision criteria for each branch would be of great help. Attempts in this direction are KADS' taxonomy of interpretation models or the distinction between class 1, 2 and 3 design in the GT approach. The differences described so far are summarized in table 4. The entries of the last row heavily depend on the knowledge acquisition tools supporting the approach.

Table 4: Major differences between the four approaches.

	GT approach	KADS	MT approach	RLM approach
Method definition language	No	Yes	No	No
Operational	Yes	No	Yes	Yes
Model of ...	System	Expert or system	System	System
Combinable	Via message passing	Yes	No	No
Model characterization	Verbal description, input/output.	Inference structure, knowledge needs, verbal descriptions.	Verbal description	Verbal description, knowledge needs.
Intended users	KE/expert	KE/expert	KE(PROTÉGÉ) and expert (p-OPAL).	Expert

4.3. Hypothesis III: *Models guide the knowledge acquisition process and structure the knowledge base*

4.3.1. The Knowledge Acquisition Facet

According to hypothesis II models of problem solving can be split into a generic and a task specific part, where the former specifies what the latter should look like. Only if a concrete method is supplied, we know the precise data structures and action types, i.e. the knowledge needs of the method to be instantiated or filled in. Even more, the elicited task-specific knowledge can only be converted into a running system if the method is supplemented by an implementation or a program generator.

In the GT approach the generic actions and data structures are object oriented definitions of the specialists, which must be instantiated by the knowledge engineer to encode an application (see fig. 2).

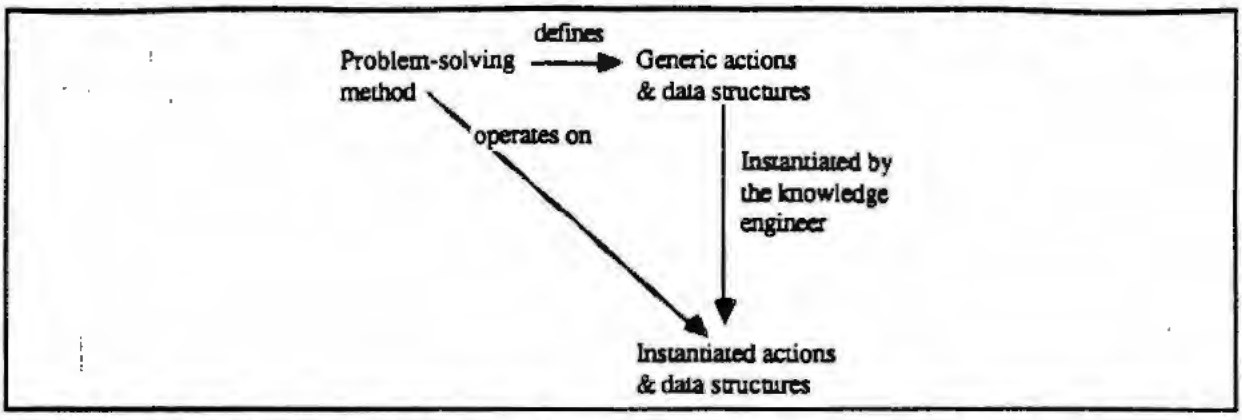


Fig. 2: The relation of method, generic actions and instances in the GT approach.

In the RLM approach generic actions are the roles of strong methods that are filled in by the specialized knowledge acquisition tools (see fig. 3), e.g. the fixes that are defined using SALT.

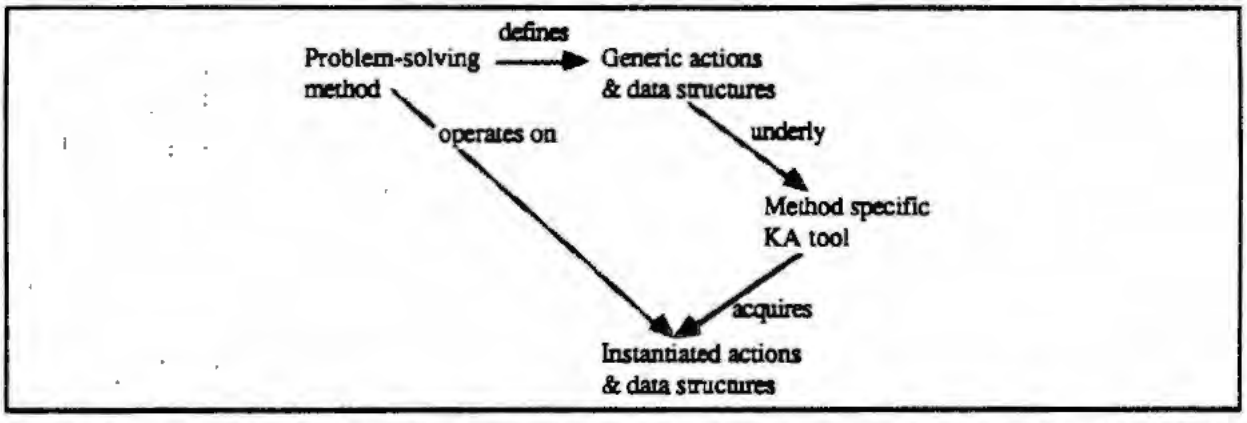


Fig. 3: The relation of method, generic actions and instantiated actions that are acquired by specialized knowledge acquisition tools (KA tools), e.g. SALT.

In the MT approach the generic actions (e.g. planning actions) are refined into task specific generic actions (e.g. chemotherapy) using PROTÉGÉ. The task specific generic actions are then instantiated using specialized editors that use information on the underlying task types (e.g. p-OPAL) to define instantiated task actions (e.g. VP-16) (see fig. 4).

Thus the MT approach makes a distinction between method specific tools that use the generic vocabulary of the method and task specific tools that use task specific refinements of the method vocabulary in order to communicate directly with the expert.

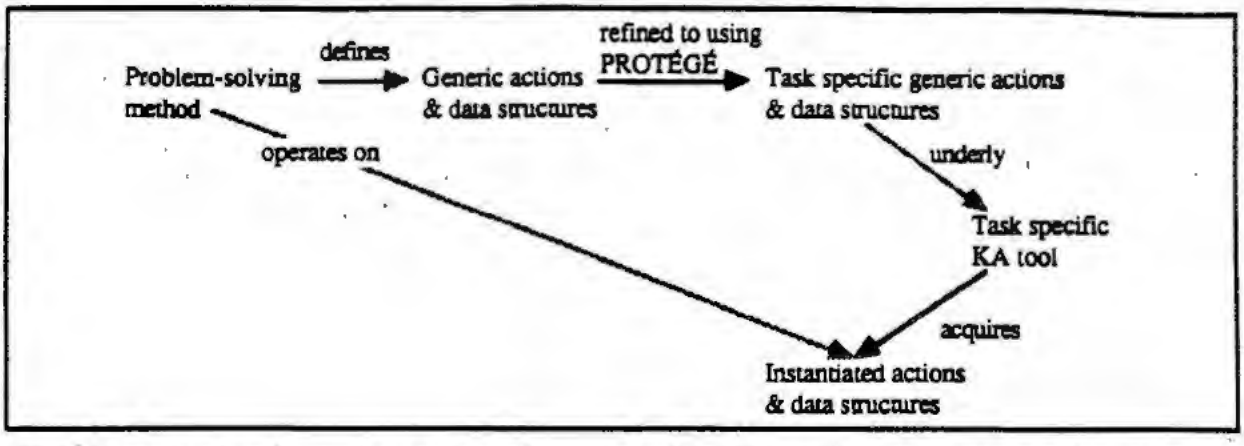


Fig. 4: The interaction of method, method specific generic actions, task specific generic actions and instances in the MT approach.

KADS models being rather abstract descriptions, they indicate the knowledge needs in terms of meta-classes and knowledge sources. These knowledge needs are not as concrete as in the other approaches which come with concrete problem solving methods. It is not impossible though, to give a very detailed description of the functioning of a KADS model which will result in the description of a method that in turn allows for the definition of generic actions and data structures.

Figure 5 summarizes the potential interaction of methods, generic actions and task or method specific knowledge acquisition tools.

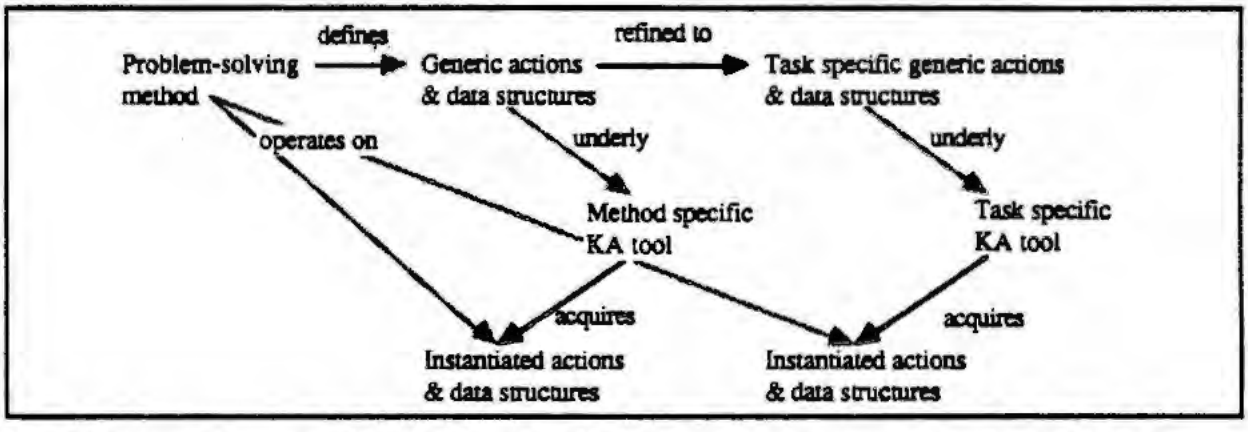


Fig. 5: The general interaction of methods, generic actions and instances in automated knowledge acquisition.

4.3.2. The Knowledge Base Structuring Facet

In the GT approach the knowledge base consists of the instantiated actions and their data structures. In the RLM approach the acquired knowledge is transformed into OPS5 knowledge bases. The instantiated actions, i.e. the role fillers are compiled into role specific rule types. These types are used to structure the operational knowledge base. In the RLM approach the method and the task specific generic actions are clearly defined in an intermediate data structure (IDS) for PROTÉGÉ and p-OPAL. They are translated into e-ONCOCIN knowledge bases. It is possible that some of the structure is lost in this process. KADS models may be arbitrarily implemented, but it is recommended to reflect the structure of the model in the knowledge base. In this case too, the model structures the knowledge base.

5. Summary

Our comparison has shown that there are major differences between the modeling approaches. It is obvious that KADS is far more flexible than other approaches, which is due to its total lack of operational semantics. The other three approaches differ mainly in their ways to represent the models in their knowledge base (GT and MT do have an internal representation, RLM does not), and in their capability to reduce the terminological gap between the expert's vocabulary and the system's knowledge needs (MT uses domain terminology, the others don't).

In the future, high flexibility is needed so that the knowledge engineer can compose, modify or design new models at need. Internal representations that explicate the problem solving models are preconditions for problem solvers with reflective capabilities that go beyond the symbol level type of reasoning. It is obvious that the use of task-specific vocabulary in the knowledge acquisition tools is a big step towards clients' maintenance of the knowledge base.

A combination of the four approaches looks very promising: the flexibility of KADS to define models, the explicit representation of the operational modeling constructs as it is done in the GT approach, the role-driven knowledge acquisition capabilities of the RLM approach, combined with editors that separate the task of the knowledge engineer from the responsibility of the domain expert, as it is done in the MT approach.

Acknowledgements

We thank Georg Klinker, Katharina Morik, Jean-Paul Krivine, Karin Lagrange, Mark Musen, Hans Voß and the members of the discussion groups at EKAW90 for their very helpful comments and encouragements.

References

Allen, 1983. Allen, J.F. Maintaining Knowledge About Temporal Intervals. *Communications of the ACM 26*, 11 (1983), 823 - 843.

Brachman, 1979. Brachman, R.J. On the Epistemological Status of Semantic Networks. In *Associative Networks: Representation of and use of knowledge by computers*. Academic Press, Findler, N.V., New York, 1979.

Breuker et al, 1987. 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. Tech. Rept. Esprit Project 1098, Deliverable Task A1, University of Amsterdam, Social Sciences Informatics, Amsterdam, 1987.

Breuker and Wielinga, 1989. Breuker, J. and Wielinga, B. Models of Expertise in Knowledge Acquisition. In *Topics in Expert System Design, Methodologies abd Tools*. North-Holland, Guida, G. and Tasso, C., pp. 265 - 295, Amsterdam, 1989.

Brown and Chandrasekaran, 1989. Brown, D.C. and Chandrasekaran, B. *Design Problem Solving: Knowledge Structures and Control Strategies*, Pitman, London , Research Notes in Artificial Intelligence(1989).

Bylander and Chandrasekaran, 1987. Bylander, T. and Chandrasekaran, B. Generic Tasks for Knowledge-Based Reasoning: The 'right' Level of Abstraction for Knowledge Acquisition. Tech. Rept. 87-TB-KNOWAC, Technical Research Report, Ohio State University , Laboratory for Artificial Intelligence Research, Columbus, 1987.

Chandrasekaran, 1986. Chandrasekaran, B. Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert* (Fall 1986), 32 - 30.

Chandrasekaran, 1988. Chandrasekaran, B. Generic Tasks as Building Blocks for Knowledge-Based Systems: the diagnosis and routine design examples. *The Knowledge Engineering Review* (Octobre 1988), 183 - 210.

Clancey, 1985. Clancey, W. Heuristic Classification. *Artificial Intelligence* 27(1985), 289 - 350.

Eshelman, 1988. Eshelman, L. MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Marcus, S., Ch. 3, pp. 37 - 80, Boston, 1988.

Forbus, 1984. Forbus, K.D. Qualitative Process Theory. *Artificial Intelligence* (1985), 85 - 168.

Karbach, Voß and Tong, 1988. Karbach, W., Voß, A. and Tong, X. Filling in the Knowledge Acquisition Gap: via KADS' Models of Expertise to ZDEST-2's Expert Systems. In *Proceedings of EKAW88*, Boose, J.H., Gaines, B. and Linster, M., GMD, St. Augustin, 1988, pp. 31/1 - 31/17.

Klinker et al, 1989. Klinker, G., Boyd, C., Dong, D., Maiman, J., McDermott, J. and Schnellbach, R. Building Expert Systems with KNACK. *Knowledge Acquisition 1*, 3 (September 1989), 299 - 320.

Kodratoff and Tecuci, 1988. Kodratoff, Y. and Tecuci, G. Learning at Different Levels of Knowledge. In *Proceedings of EKAW88*, Boose, J.H., Gaines, B.R. and Linster, M., GMD, St. Augustin, 1988, pp. 3/1 - 3/17.

Kuipers, 1984. Kuipers, B. Commonsense Reasoning about Causality: Deriving Behavior from Structure. *AI Journal* (1984), 169 - 203.

Marcus, 1988. Marcus, S. A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers, Marcus, S., Ch. 4, pp. 81 - 124, Boston, 1988.

Marcus, 1989. Marcus, S. Understanding Decision Ordering from a Piece-Meal Collection of Knowledge. *Knowledge Acquisition 1*, 3 (September 1989), 279 - 298.

McDermott, 1988. McDermott, J. Preliminary Steps Toward a Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic, Marcus, S., Ch. 8, pp. 225 - 256, Boston, 1988.

Morik, 1987. Morik, K. Acquiring Domain Models. *International Journal of Man-Machine Studies* 26(1987), 93-104.

Musen, 1988. Musen, M.A. Conceptual Models of Interactive Knowledge Acquisition Tools. In *Proceedings of EKAW88*, Boose, J.H., Gaines, B.R. and Linster, M., GMD, St. Augustin, 1988, pp. 26/1 - 26/15.

Musen, 1989. Musen, M.A. *Automated Generation of Model-Based Knowledge Acquisition Tools*. Pitman Publishing, London , Research Notes in Artificial Intelligence(1989).

Musen et al, 1987. 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.

Norman, 1983. Norman, D. Some Observations on Mental Models. In *Mental Models*. Hillsdale, Genmer, D., pp. 7 - 14, New Jersey, 1983.

Schreiber et al, 1987. Schreiber, G., Bredeweg, B., Davoodi, M. and Wielinga, B. Towards a Design Methodology for KBS. Tech. Rept. 97, ESPRIT Project 1098, Deliverable D8, VF Memo , University of Amsterdam, Amsterdam, 1987.

Schreiber et al, 1988. Schreiber, G., Breuker, J., Bredeweg, B. and Wielinga, B. Modelling in KBS Developement. In *Proceedings of EKAW88*, Boose, J.H., Gaines, B.R. and Linster, M., GMD, St. Augustin, 1988, pp. 7/1 - 7/15.

Voß and Linster, 1989. Voß, H. and Linster, M. Interval-Based Envisioning in HIQUAL. *Applied Artificial Intelligence, Special Issue on Causal Modeling 2-3*(1989), 17 - 43, Also published in: W. Horn(ed.):Causal AI Models - Steps Toward Applications, Hemisphere Publ., New York, 1990, p. 17 - 43.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Usable and Reusable Programming Constructs

Georg Klinker, Carlos Bhola, Geoffroy Dallemagne, David Marques, and John McDermott

Digital Equipment Corporation
111 Locke Drive
Marlboro, MA. 01752

Our research goal is to make programming easier. A central focus in our research is developing programming constructs, called mechanisms, that are both (1) usable and (2) reusable. A mechanism is usable if it can be used to automate a 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 domains and tasks. Focusing on usability can lead to application-specific mechanisms, that is, can reduce reusability. On the other hand, focusing on reusability can lead to task-independent structures, that is, lessen their usability. The trick is to come up with mechanisms satisfying both requirements: usability and reusability. We have identified some quantitative characteristics describing such mechanisms.

1 Introduction

Making programming easier has been -- and still is -- the central focus of many research efforts. These efforts generally target either the expert programmer or non-expert programmers. Efforts which target the expert programmer are under way in software reusability [for example, Freeman 87, Krueger 89] and programming assistance [for example, Rich 90]. A central focus of those efforts are programming constructs that are reusable. A programming construct is reusable if it can be employed for several domains and tasks. Figure 1 defines the interdependencies between task and domain.

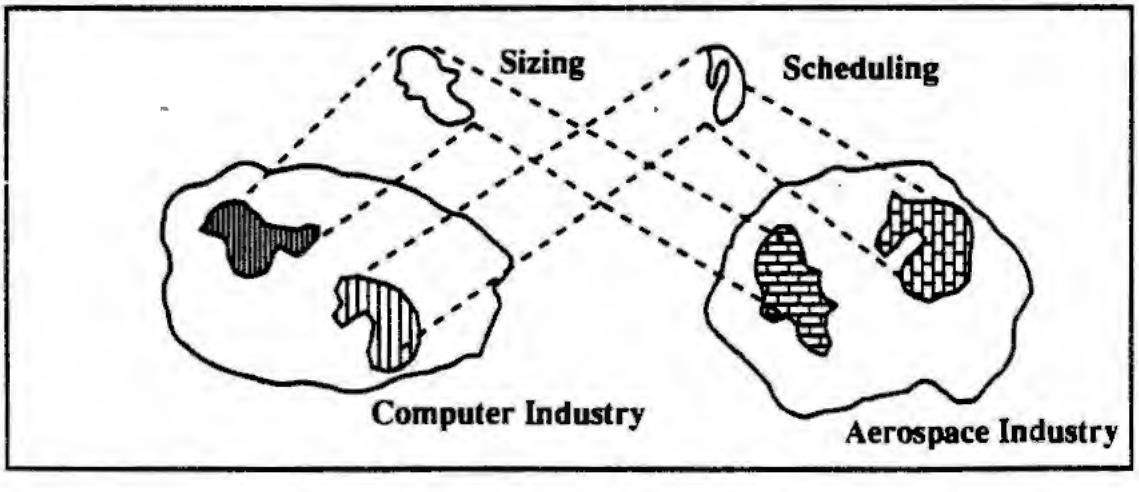


Figure 1 Task, Domain, and Application

A task, for example sizing and scheduling, is defined as the set of processes used to perform an activity. A task can be performed in different domains. A domain is comprised of the data operated upon by a process. In Figure 1, computer and aerospace industries are examples of domains. The projection of a task onto a domain is called an application (or task instance). For example, scheduling the delivery of computer hardware and scheduling the manufacturing process for airplanes are applications defined by the projection of the scheduling task onto the computer and aerospace domains.

Other research efforts are directed at making programming easier for non-expert programmers. Automated knowledge acquisition [for example, Bennett 85, Birmingham 88, Boose 84, Clancey 83, Davis 79, Eshelman 88, Klinker 88, Marcus 88, Musen 89] and knowledge-level analysis [for example, Newell 81] contribute to that goal. Here, the central focus is to develop usable programming constructs. A programming construct is usable, if a person who understands and performs a task, but does not necessarily know how to program, can use it to automate parts of the task.

Programming constructs suggested by those research efforts are either close to the problem, that is, their scope is very narrow, or close to the implementation -- that is, they do not make programming easier for non-expert programmers. A central focus in our research is developing programming constructs, called mechanisms, that are both (1) usable and (2) reusable. A mechanism is usable if it can be used to automate a 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 domains and tasks. Focusing on usability can lead to application-specific mechanisms, that is, can reduce reusability. On the other hand, focusing on reusability can lead to task-independent structures, that is, lessen their usability. The trick is to come up with mechanisms satisfying both requirements: usability and reusability. Current research with closely related goals includes [Breuker 89, Chandra 83, Malone 88, Roth 90, Yost 89].

We are developing a framework that exploits usable and reusable mechanisms. A high level overview of the framework is presented in section 2. A developer can use the framework to create an application system. A developer is a person who understands and performs a task, but might not be a programmer. The developer uses the framework to identify mechanisms relevant to his task, combine them, and generate, test, and refine an application system. Section 3 demonstrates a developer creating a scheduling application. In turn, the framework imposes constraints on mechanisms. The constraints can be used to characterize mechanisms. Section 4 identifies the constraints we have identified so far and quantifies them. Those measures present the current status of our research and are derived from currently implemented mechanisms. Section 5 contains more detailed data on the existing mechanisms.

2 An Integrated Framework for Making Programming Easier

Our research is based on the assumption that a developer can describe a task in domain-specific terms but probably cannot describe the computational mechanisms a computer program must use to perform the task. Given this assumption, we hypothesize that computational mechanisms can be defined at a level of abstraction such that they can be indexed by the developer's task description, configured into a problem-solving method, and exploited for knowledge-acquisition purposes. We are developing an integrated framework for validating our hypothesis. The framework is summarized in Figure 2.

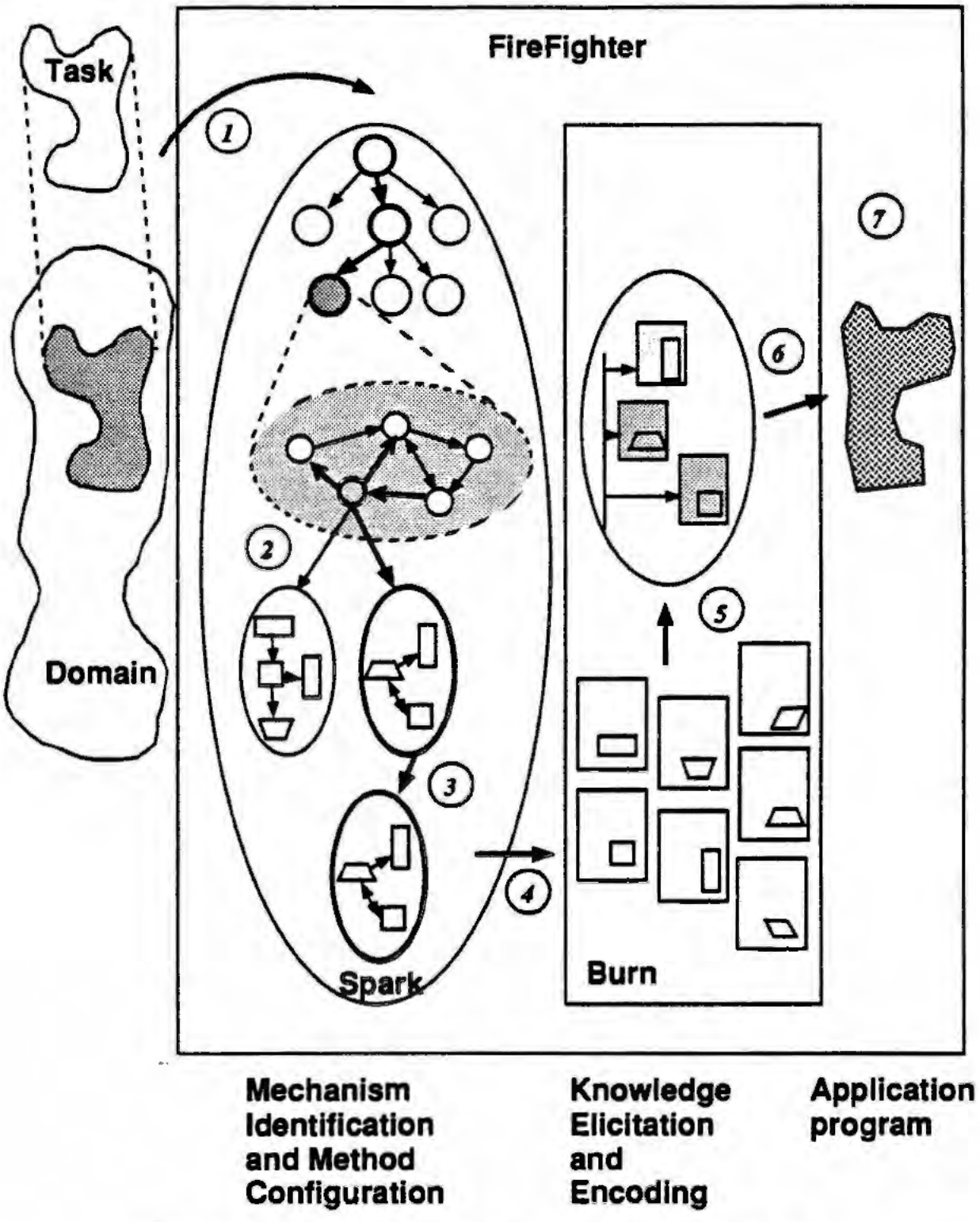


Figure 2 An Integrated Framework for Making Programming Easier

Assume that a developer wants to automate a given task. (1) He interacts with Spark to describe the task. For this purpose, Spark provides a general model of the developer's domain. The domain model describes possible activities, resources consumed by those activities, products produced, and agents performing those activities to the level of detail relevant to the devel-

oper. The developer identifies the activities, resources, and products relevant for the task he wants to automate. (2) Associated with each activity in the model are configurations of mechanisms automating parts or all of the activities. Each mechanism configuration could automate the activities but presupposes different informational requirements.

(3) Spark's next task is to differentiate among the candidate configurations. It asks a set of questions to verify the availability of the information required by the mechanisms and to identify relevant task features. Spark then selects the configuration most compatible with those features.

If the selected mechanism configuration requires input not available to the problem-solver, Spark identifies additional mechanisms which produce that input. This process continues until Spark has identified a collection of mechanisms that, together with the selected mechanism configuration, use the available input to produce the required output. Spark then integrates the additional mechanisms with the selected configuration and this results in a problem-solving method. The problem-solving method is a configuration of mechanisms solving the developer's task.

(4) This configuration of mechanisms is then passed to Burn which manages a collection of specialized knowledge acquisition tools. Each mechanism in Spark's library has a corresponding knowledge-acquisition tool in Burn's library. Spark's configuration of mechanisms thus uniquely identifies the knowledge-acquisition tools required to elicit and encode the knowledge needed to perform the task. (5) Each knowledge-acquisition tool interacts with the developer to create a knowledge base for its mechanism. (6) This knowledge base and the pre-defined code implementing the mechanisms form a specialized prototype system that automates the application task.

(7) Since both Spark and Burn deliberately use heuristics to configure the problem-solving method and to elicit the domain knowledge, it is to be expected that the first version of the application program will not solve the task to the developer's satisfaction. Thus, the developer uses a debugger, FireFighter, to revise the application program. There are two reasons why a mismatch between an application program and a task is likely: (1) Burn did not elicit enough knowledge, (2) the mechanism configuration was not appropriate for the application task. FireFighter assists the developer in determining and resolving the problem. It can be used to acquire more knowledge, replace a mechanism configuration or an individual mechanism together with the associated knowledge acquisition tools, and/or change a mechanism and the associated knowledge acquisition tool. The refinement of the application program is an iterative process that ends when the developer is satisfied with the program's performance.

3 Automating a Scheduling Application

This section provides an example of an interaction with Spark and Burn. Since we are in the process of implementing FireFighter, a description of that tool is not available at this time. In this example a developer would like to automate the scheduling of the production of composite materials used in airframes. Figure 3 describes this manufacturing process: Raw materials are stored in a freezer and must thaw before they can be processed. Once thawed, the material must be processed within a fixed amount of time. The thawed material is "layed-up" on forms, called bond molds. There are a limited number of bond molds available. The layed-up material is then put on dollies and a coating is applied to hold the different layers together. There are a

limited number of dollies available. This is called the "bagging" process. The bagged material is then cured in an oven, called an autoclave. There are a limited number of autoclaves available. The curing process results in the composite material.

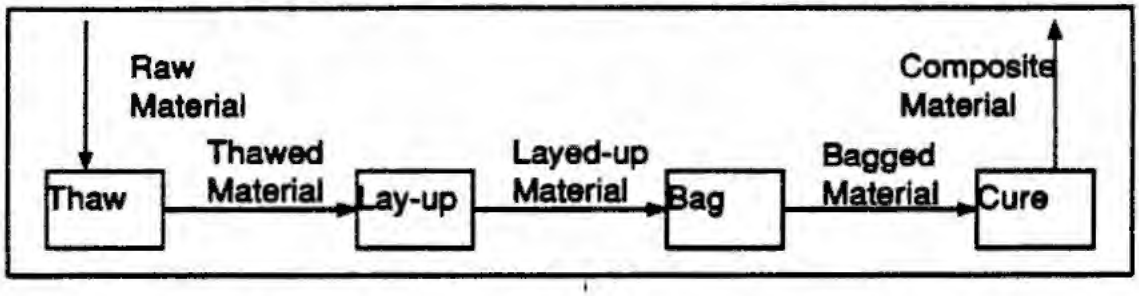


Figure 3 The Manufacturing of Composite Materials Used in Airframes

The developer interacts with Spark, Burn, and FireFighter to create a scheduling program, Scheduler, for the manufacturing process described above. The first step is to describe the task to Spark. Since Spark communicates in task- and domain-specific terms with the developer, it first asks the developer to identify the industry relevant to the task. Figure 4 demonstrates the developer identifying "discrete manufacturing" as the relevant industry type.

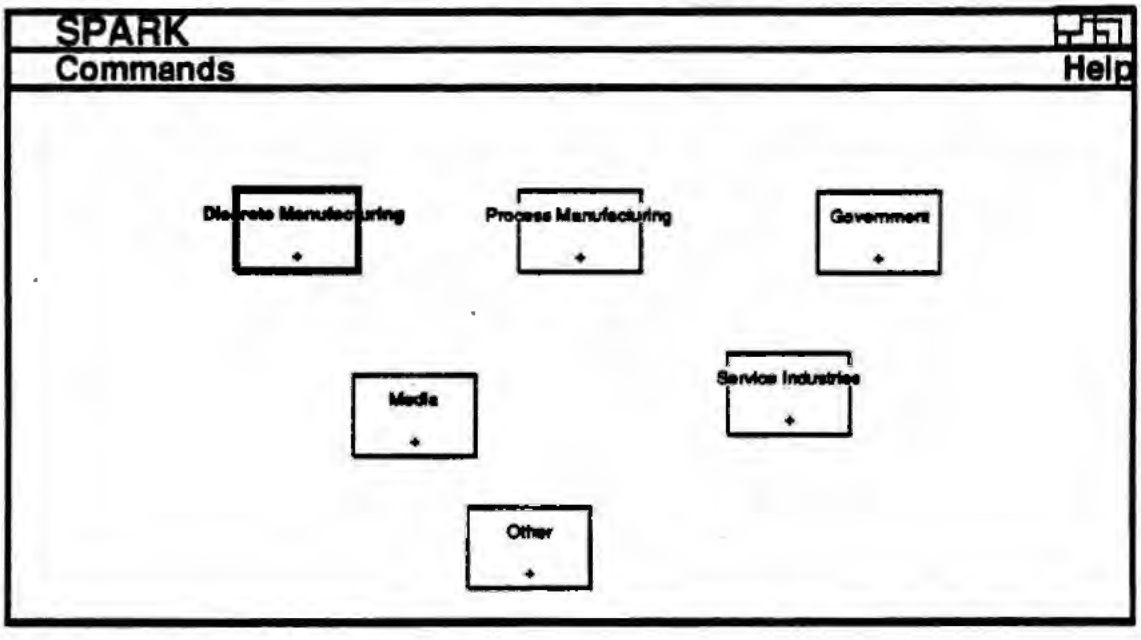


Figure 4 Determining the Industry Type

Spark uses the industry type as a first index into an extensive model, called the enterprise model, of activities in a comprehensive set of domains. The developer is presented with a high level view of activities within discrete manufacturing and is asked to locate the activities which best describe the task. A more specialized view of each activity is available to the developer as required. This refinement process continues until a level of detail is reached that the developer recognizes as a description of the task. For the purposes of this example, the developer is satisfied with the description shown in Figure 5.

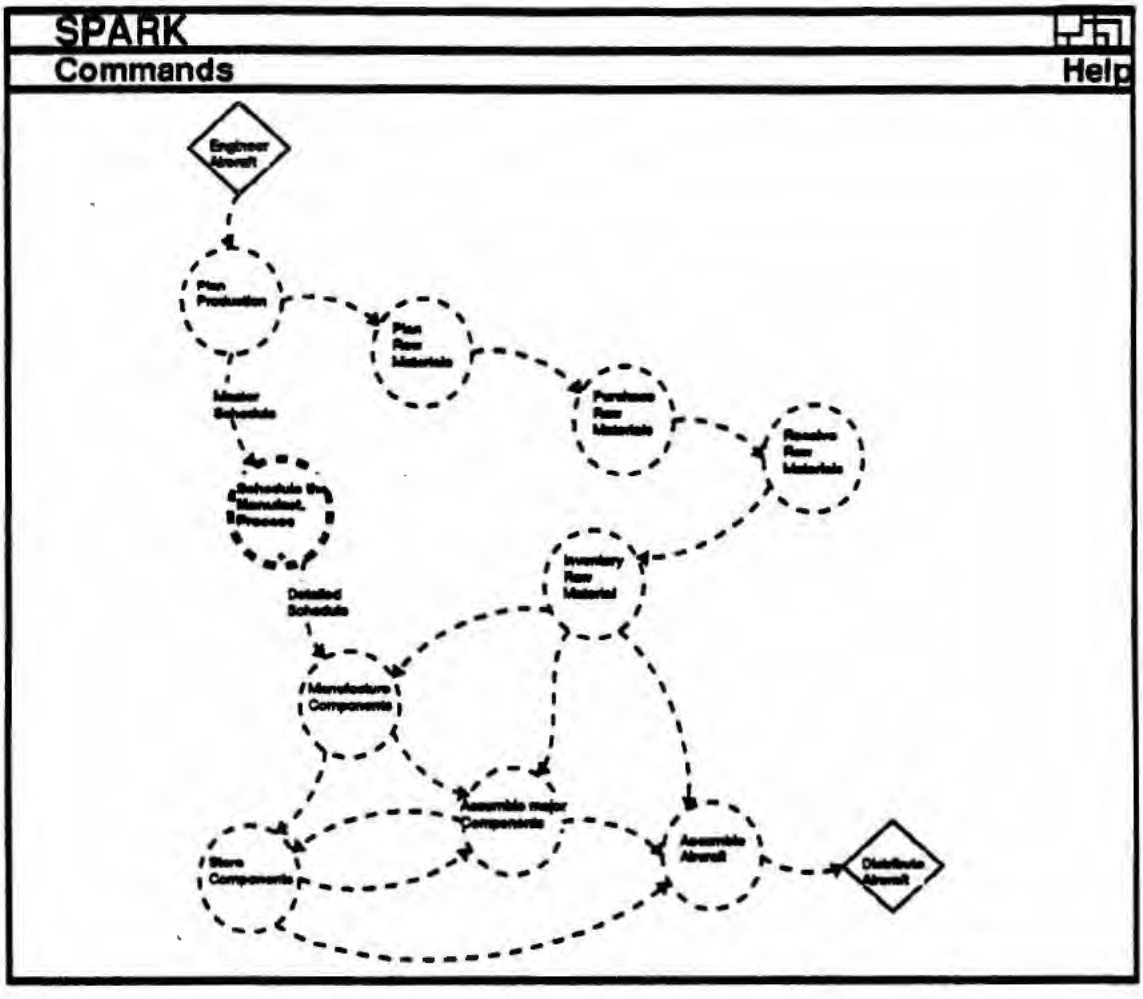


Figure 5 The Definition of the Scheduling Task

Activities are represented by dashed circles and the resources needed for, or produced by, each activity are represented by dashed arrows. The bold circle indicates the activity selected by the developer. Associated with the selected activity are mechanism configurations. A mechanism configuration is a set of mechanisms automating the selected activities. Figure 6 shows the

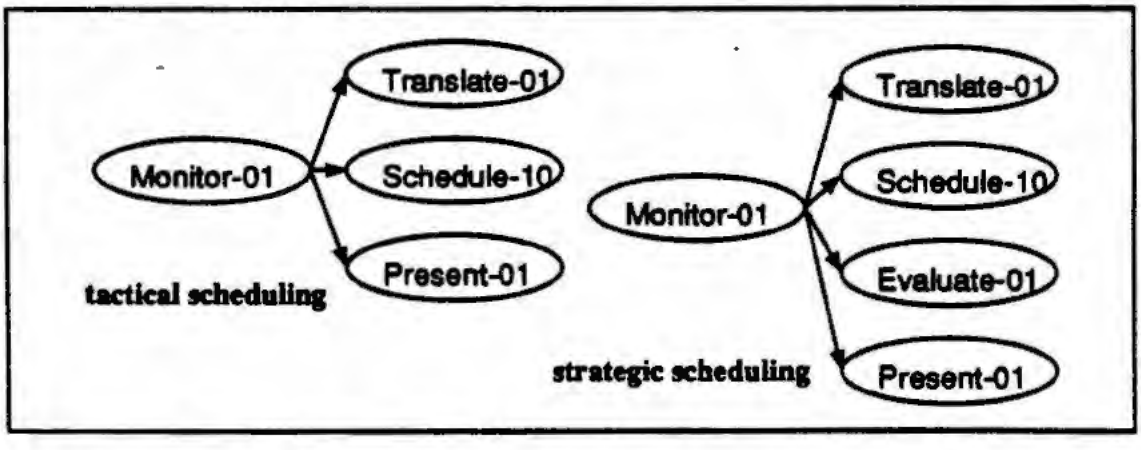


Figure 6 Candidate Mechanism Configurations for the Scheduling Task

candidate configurations for the scheduling task. Mechanism configurations as well as mechanisms are internal to Spark. They are not presented to the developer, since he need not be concerned with the computational aspects of the task.

The "strategic scheduling" configuration describes a program that can be used to plan the scheduling approach. Monitor-01 implements the Scheduler's interface. Translate-01 reads orders from a file. Schedule-10 produces an actual schedule. Evaluate-01 allows the user to evaluate the produced schedule from different perspectives, eg, costs and utilization. It further allows for specializing Schedule-10 and Present-01 for different scheduling approaches Finally, Present-01 presents the schedule to the user in the form of lists and charts.

The "tactical scheduling" configuration describes an application deployed on the manufacturing floor for day-to-day use. It uses a subset of the mechanisms from the "strategic scheduling" application. In this case, Translate-01 reads the orders from a database.

Once the developer identified the task in the enterprise model, Spark presents more detail in an attempt to acquire some of the knowledge used to instantiate the mechanisms. Figure 7 demonstrates a more detailed definition of the scheduling task.

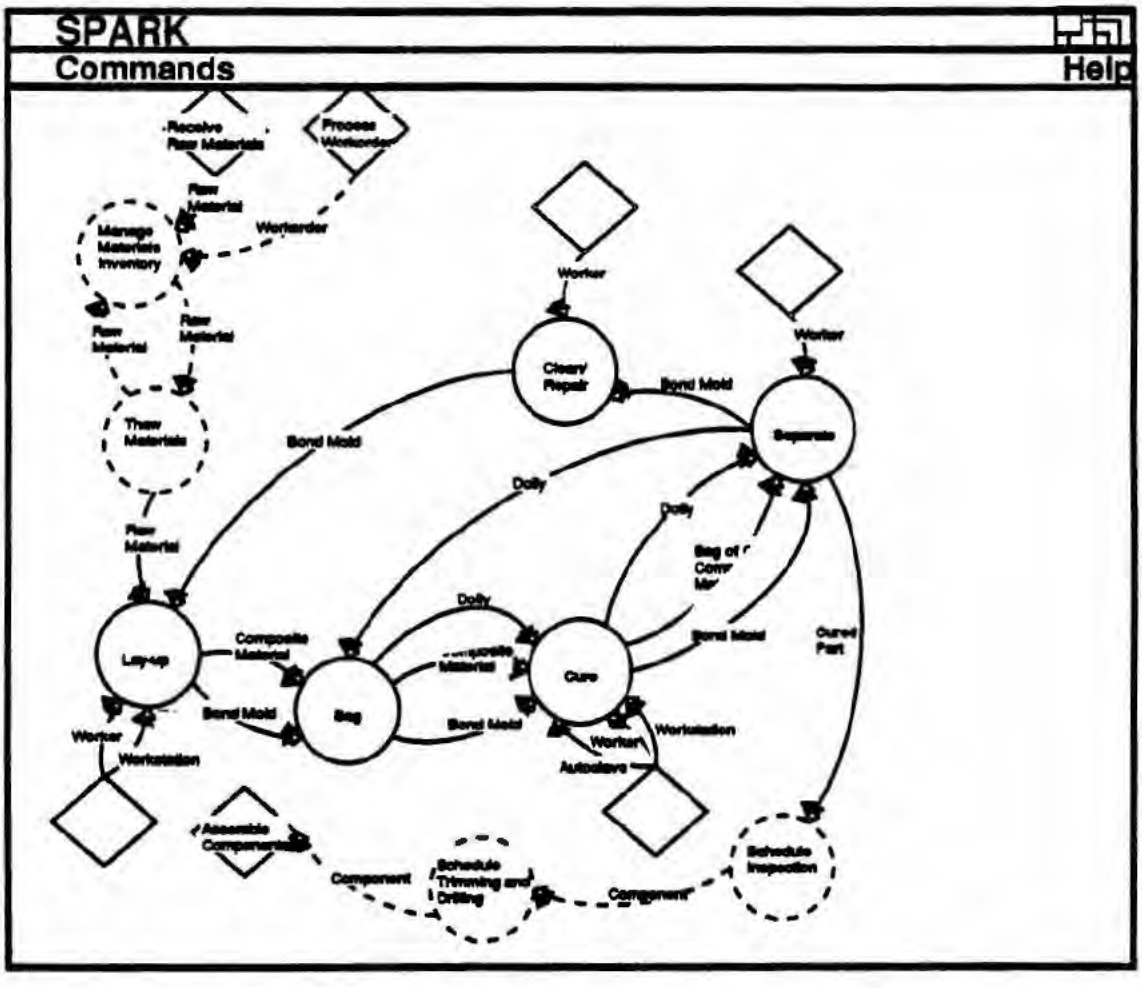


Figure 7 More Detail about the Scheduling Task

The solid circles and arrows define the activities that were selected by the developer as part of the task.

Since there is more than one mechanism configuration associated with the activities selected by the developer, Spark now asks questions, like that in Figure 8, to resolve this ambiguity.

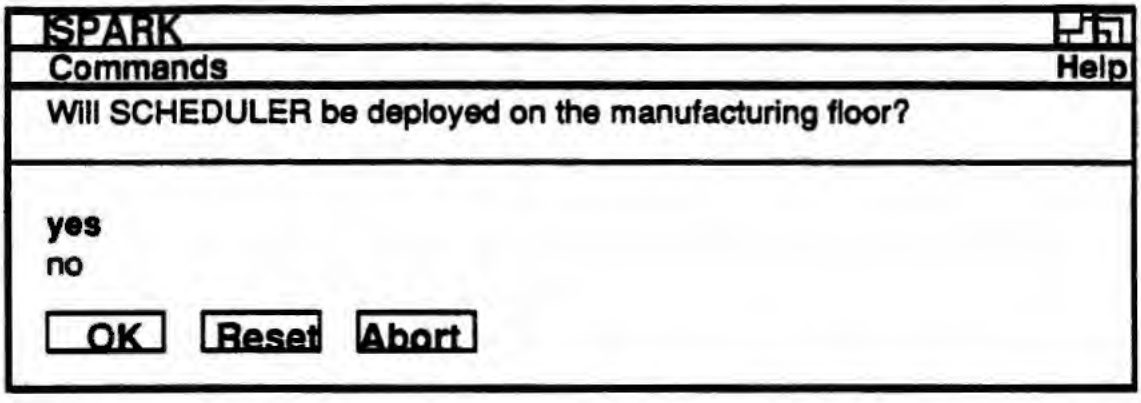


Figure 8 Differentiating Between Mechanism Configurations

Based on the developer's answer Spark rejects the "strategic scheduling" configuration, confirms the "tactical scheduling" configuration (see Figure 6), and passes that configuration to Burn.

Burn elicits the knowledge from the developer that customizes a mechanism configuration for the problem at hand. It manages a library of knowledge-acquisition tools, each of which is associated with one of Spark's mechanisms. Burn invokes the knowledge-acquisition tools relevant to the mechanisms of the selected mechanism configuration. Each knowledge-acquisition tool is specialized for its mechanism. For example, the knowledge-acquisition tool for Schedule-10 acquires the knowledge describing the products, activities, resources, and constraints on how to associate resources with tasks. Figure 9 illustrates the developer defining the "bond mold" resources.

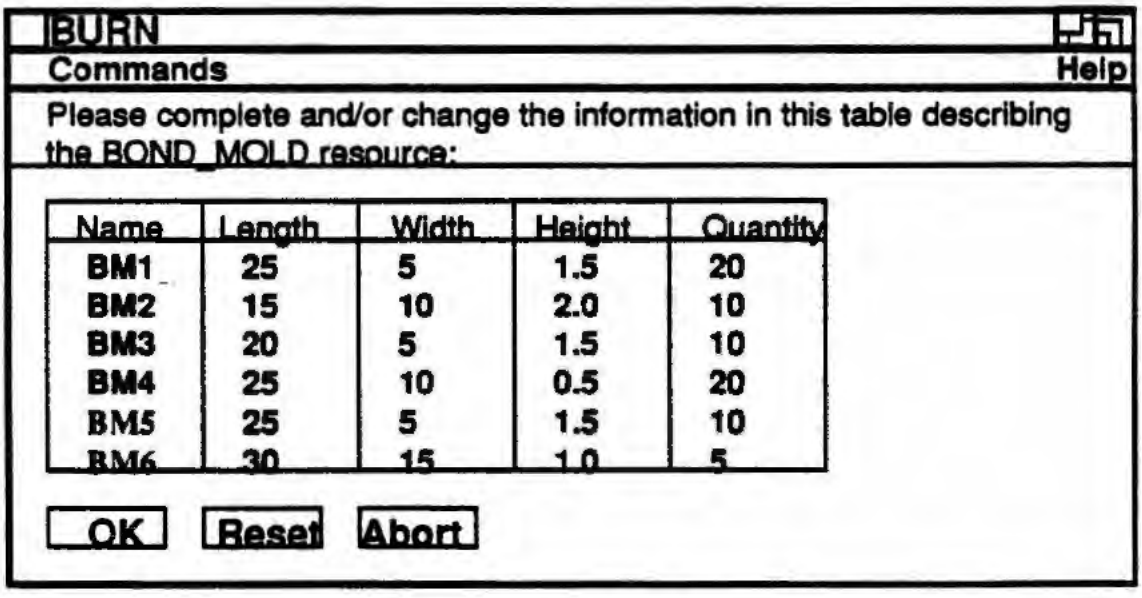


Figure 9 Acquiring Resource Knowledge

The context for this question was provided by Spark. The enterprise model, shown in Figure 7,

identified "bond molds" as a relevant resource. It further defined spatial constraints as important.

The result of a developer's interaction with Burn is an application program, Scheduler. Burn compiles it and FireFighter runs Scheduler. Figure 10 shows a part of a schedule produced by Scheduler.

SCHEDULER

Commands Help

Schedule for Period 3:

Order	Due-Date	Task	Start	End	Resources
P1	25	Lay-up	1	4	BM1, W1
		Bag	8	9	D1, W2, W3
		Cure	12	15	AC1, W4
		Separate	15	16	W5
		Clean/R	16	17	W6
P3	30	Lay-up	4	8	BM1, W1

OK Reset Abort

Figure 10 A Schedule Produced by Scheduler

FireFighter then asks the developer whether Scheduler performed adequately. For the purposes of this example the developer was satisfied with the solution produced by Scheduler and terminates his interaction with the framework. If the developer was not satisfied with Scheduler's results, he would use FireFighter and refine the application program.

4 Usable and Reusable Programming Constructs

A central focus of our research is developing programming constructs that are (1) usable and (2) reusable. These requirements are inversely related: in trying to make a mechanism usable for a non-programmer we are making it less reusable and by increasing a mechanism's reusability we make it less usable. By emphasizing usability we lean towards developing mechanisms that exert strong expectations on the kinds of tasks and domains in which they can be used In exploiting those expectations a mechanism gains usability, but tends to lose reusability. In general, the usability requirement leads to application-specific mechanisms

By emphasizing reusability we lean towards developing mechanisms that impose very little constraints on the kinds of tasks and domains they can be used for. By not having to pay attention to those constraints a mechanism gains reusability, but tends to become less usable. In general, the reusability requirement leads to task and domain independent mechanisms.

We have identified some of the issues involved in defining a mechanism's usability and reusability requirements:

- **Indexing issue:** Mechanisms are identified by the developer's knowledge of his task. That is, the mental model of the developer must be mapped to Spark's computational model.
- **Instantiating issue:** Mechanisms can be viewed as skeletal plans which can be applied across a range of tasks and domains. They must be instantiated into application-specific mechanisms.
- **Configuration issue:** Each mechanism implements some aspects of a developer's task. Therefore, mechanisms must be composed into a mechanism configuration to produce a complete solution.

The following explores these issues, points out their impact on the usability and reusability requirements for mechanisms, and presents some quantitative data that characterize mechanisms. These measures present the current status of our research and are derived from the currently implemented mechanisms. They are preliminary and we expect that they will change over the next one or two years. Figure 11 provides a summary of the current project status.

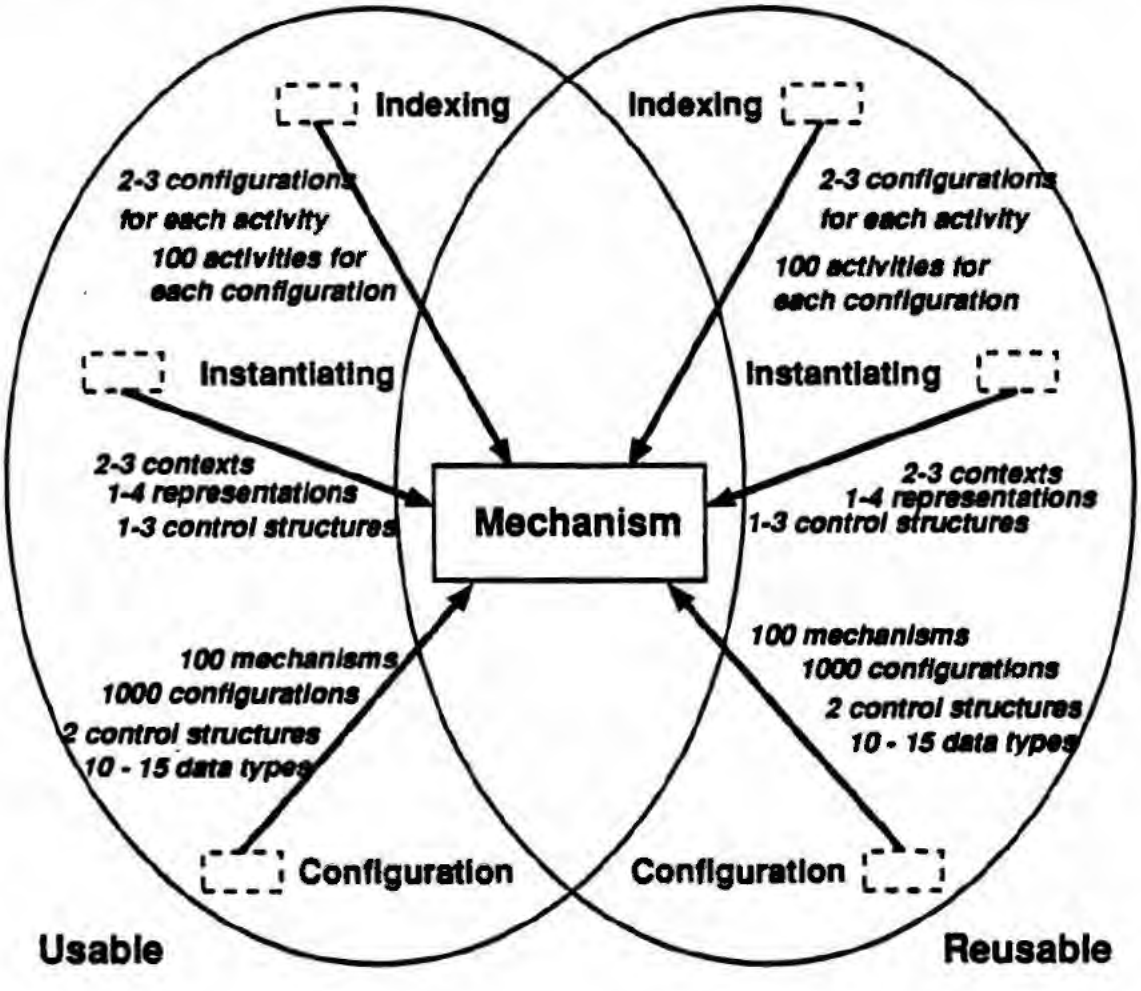


Figure 11 : Some Constraints on Usable and Reusable Mechanisms

The numbers are detailed in section 5.

4.1 Indexing

Mechanisms are identified by a developer's knowledge of a task That is, the mental model of the expert must be mapped to a computational model. This problem is similar in nature and difficulty to the communication problem between two people who are from very different cultural backgrounds and who speak different languages: a developer, who is non-programmer, must describe a task to a computer program; at the same time, a computer program must understand the developer's problem.

In our work, the enterprise model represents the developer's view of tasks. It describes the developer's task in terms of activities and their resources using the developer's concepts and terminology. The activities are presented at different levels of detail. At any level, the developer can customize the model by deleting and/or adding activities and resources and by changing the terminology. Figure 7 provides an example of an enterprise model customized for the manufacturing of composite materials used in airframes.

The library of mechanisms and associated mechanism configurations defines Spark's language. A mechanism includes source code that can be compiled, together with other mechanisms, into a computer program. It also includes questions that can be used to resolve ambiguities when the developer's task definition results in multiple configurations. To date, the mapping between activities in the enterprise model and mechanisms in Spark's library is "handcrafted". The mechanism creator explicitly defines the mappings when placing a new mechanism in Spark's library. The actual mapping is from an activity to a configuration of mechanisms implementing the activity The questions that disambiguate between multiple configurations are also "handcrafted".[1]

By emphasizing usability we lean towards developing mechanisms that are specific to an activity described in the enterprise model. This leads to a one-to-one mapping from activities to mechanism configurations: a mechanism is only used in the implementation of one activity. By emphasizing reusability we lean towards developing mechanisms that can be used to implement a broad range of activities described in the enterprise model. This leads to a complex mapping from activities to mechanism configurations: the same mechanism is used in the implementation of every activity. These considerations lead to some guidelines for defining mechanisms.

- Define mechanisms in a way that a small number (currently 1 or 2, see figure 17) of configurations will cover most of the significantly different computational alternatives for an activity. In other words, define mechanisms such that for almost any variant of a particular activity, one out of a small number of configurations will be close to appropriate. For example, Figure 6 indicates that 2 configurations are used to implement the scheduling of composite materials production. Our bet is that the number of configurations for an activity will change to 2 or 3.
- Define mechanisms in a way that a configuration will cover similar computational alternatives for a reasonable number (currently 1 or 2, see figure 17) of significantly different activities. In other words, define mechanisms such that they can be specialized for different

[1] Some guidelines for defining questions exist: answers to questions reject or confirm a mechanism configuration; a question must be specific to either a configuration or an individual mechanism; furthermore, a question must take advantage of the context provided by an activity.

activities. Our bet is that the number of activities implemented by the same configuration will increase to the order of 100.

4.2 Instantiating

Mechanisms can be viewed as skeletal plans which can be applied across a range of tasks and domains. They have some task specific and some domain specific knowledge but are not situation specific. The knowledge-acquisition tool associated with a mechanism instantiates a mechanism into a situation-specific mechanism. The knowledge-acquisition tool gains power by exploiting presupposed control structures and representations. An example of instantiating a mechanism is demonstrated in Figure 9. The extent to which a knowledge-acquisition tool presupposes control structures and representation contributes to the usability and reusability of its corresponding mechanism.

By emphasizing usability we lean towards developing mechanisms that impose strong constraints on the kinds of knowledge they can use, how to represent that knowledge, and how to acquire it. By emphasizing reusability we lean towards developing mechanisms that that impose little constraints on the kinds of knowledge they can use, how to represent that knowledge, and how to acquire it. These considerations lead to some guidelines for defining mechanisms:

- Presuppose a small number (currently 1 - 3, see figure 18) of control structures. By presupposing multiple control structures a knowledge-acquisition tool can be used to customize its mechanism for different tasks. For example, the Translate-02 mechanism can be used to generate SQL or UIL code. Its knowledge-acquisition tool asks a simple question to instantiate the mechanism with a control structure that will either customize the mechanism to interface with an RDB database or the DECwindows environment. Our bet is that the current number of control structures is about right.
- Presuppose a small number (currently 1 - 4, see figure 18) of representations. By presupposing multiple representations a knowledge-acquisition tool can be used to customize its mechanism for different domains. For example, the Schedule-10 mechanism is applicable for a variety of resource types. A "material" resource is consumed by a task whereas a "form" resource is used to group multiple orders for a limited amount of time. Both resource types are represented differently. The Schedule-10 mechanism can deal with both and, therefore, can be used for domains that require either one or both resource types. Its knowledge-acquisition tool asks some simple questions to determine the relevant resource types and customizes the Schedule-10 mechanism accordingly. Our bet is that the current number of representations is about right.
- Develop mechanisms which can take advantage of a small number (currently 1 or 2, see figure 18) of different contexts provided by the enterprise model. The developer can use the enterprise model to define a task at different levels of detail. If the task definition was at a fairly high level, the knowledge-acquisition tool has to acquire all the details to instantiate the mechanism. On the other hand, a fairly detailed task definition contributes to the customization of a mechanism. For example, the details about the bond mold resources can either be acquired by SCHEDULE-10's knowledge-acquisition tool (see figure 9) or is provided by the enterprise model. Our bet is that the number of different levels of details provided by the enterprise model will change to 2 or 3.

4.3 Configuration

The mechanism configurations associated with an activity in the enterprise model are composed of multiple mechanisms, each implementing a part of the activity. To date, the configuration of mechanisms is "handcrafted". The mechanism creator explicitly defines them when he adds a new mechanism to Spark's library. However, the configurations are not static. A configuration includes a list of alternatives for its individual mechanisms. If a selected configuration was not appropriate to solve the developer's task, replacing a mechanism with one of its alternatives is one of FireFighter's remedies to the problem.

By emphasizing usability we lean towards developing mechanisms which impose strong constraints on how they can be configured with other mechanisms. This leads to a very large number of different configurations. By emphasizing reusability we lean towards developing mechanisms which impose few constraints on how they can be configured with other mechanisms. This makes the actual configuration task very difficult. These considerations lead to some guidelines for defining mechanisms:

- Develop mechanisms which "hide" complicated control structures. To date, we only allow 2 ways to configure mechanisms: a mechanisms is either "appended to" and/or "nested within" other mechanisms. We bet that we will not need additional ways to combine mechanisms.
- Develop mechanisms that use pre-defined data structures for passing data. To date, those data structures are sets, lists, and trees. A mechanism can transform these external structures in its own internal representation. We bet that we will end up with 10 to 15 different data structures.
- Define mechanisms in a way that a reasonable number (currently 7, see figure 14) of configurations will be close to appropriate to implement most of the activities described in the enterprise model. In other words, define mechanisms in a way that FireFighter will be able to make any almost appropriate configuration appropriate with the addition or replacement of 1 or 2 alternative mechanisms. We bet that the number of configurations needed will be on the order of 1000 and that the number of activities described in the model will be on the order of 100000.
- Define mechanisms in a way that a reasonable number (currently 15, see figure 14) of mechanisms can be combined into different configurations. On average, a configuration consists of 4 mechanisms. We bet that the number of unique mechanisms will be on the order of 100.

5 Current Status

Our approach to defining mechanisms is empirical. We create application programs, decompose them into mechanisms, and integrate the mechanisms with the framework. This research effort will continue for some time to come. In this section, we present the data which is the basis of our current guidelines for defining mechanisms. The data must be viewed as preliminary results that point us to what to look for when we create a new mechanism.

Figures 12 and 13 contain a brief textual description of the application systems we have imple-

Candlestick, V1: 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.

Candlestick, V2: 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, and in addition, to estimate what application software is needed by a client's organization.

- Desired results: a list of computer hardware, system software, and suggested application software products.
- Data available: a detailed description of the client's organization, the activities performed within the client's organization, including: the number of personnel in each job classification (such as traders, programmers, managers and secretaries); the informational needs of each activity, the activities of each job function, the industry in which the organization exists (eg, finance, publishing, etc.), special needs such as reconfigurable user interfaces; and a database of software product descriptions.
- Expertise available: knowledge of successful existing selections, knowledge of constraints pertaining to the selection of specific software.

Softsel: The task is to select application software corresponding to the needs of a client's organization.

- Desired results: a list of application software products appropriate to a client's needs.
- Data available: a detailed description of the client's organization, the activities performed within the client's organization, including: the number of personnel in each job classification (such as traders, programmers, managers and secretaries); the informational needs of each activity, the activities of each job function, the industry in which the organization exists (eg, finance, publishing, etc.), special needs such as reconfigurable user interfaces; and a database of software product descriptions.
- Expertise available: knowledge of successful existing selections, knowledge of constraints pertaining to the selection of specific software, knowledge of how to extrapolate from existing selections.

Layout: The task is to create and layout on a workstation's screen rectangular tiles corresponding to an individual's data needs based on his view of his organization.

- Desired results: a tiled workstation screen corresponding to an individual's data needs based on his view of his organization.
- Data available: a model corresponding to the user's data needs based on his view of his organization, and a description of the workstation's screen.
- Expertise available: heuristics for placing tiles on the screen, topological knowledge to constrain tile placement on the screen, heuristics for refining tile placements to accommodate the constraints, and knowledge of how to generate UIL code.

Retriever: The task is to generate SQL 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.
- 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.
- Expertise available: heuristics for aggregating the data and knowledge of how to generate SQL code.

Figure 12 Textual Description of Application Systems -- Part 1

mented so far.

Scheduler: The task is to schedule the manufacture of composite materials used in airframes.

- Desired results: a schedule that satisfies the relevant constraints and a way to presenting relevant information about that schedule to interested parties.
- Data available: the orders and their due dates and questions from various levels of management.
- Expertise available: expertise describing the products (types of orders) and resources, expertise constraining how specific resources can be associated with particular products, and expertise on evaluating the schedule.

Spark: The task is to determine which set of computational mechanisms can collectively solve a given task and to configure those mechanisms into a problem-solving method.

- Desired results: a configuration of computational mechanisms appropriate for performing the specified task.
- Data available: a knowledge-level description of the specified task.
- Expertise available: a set of computational mechanisms which are appropriate for performing various tasks and heuristics for determining which of the set of computational mechanisms are needed to automate the task.

Figure 13 Textual Description of Application Systems -- Part 2

Figure 14 relates the application systems to the mechanisms they use.

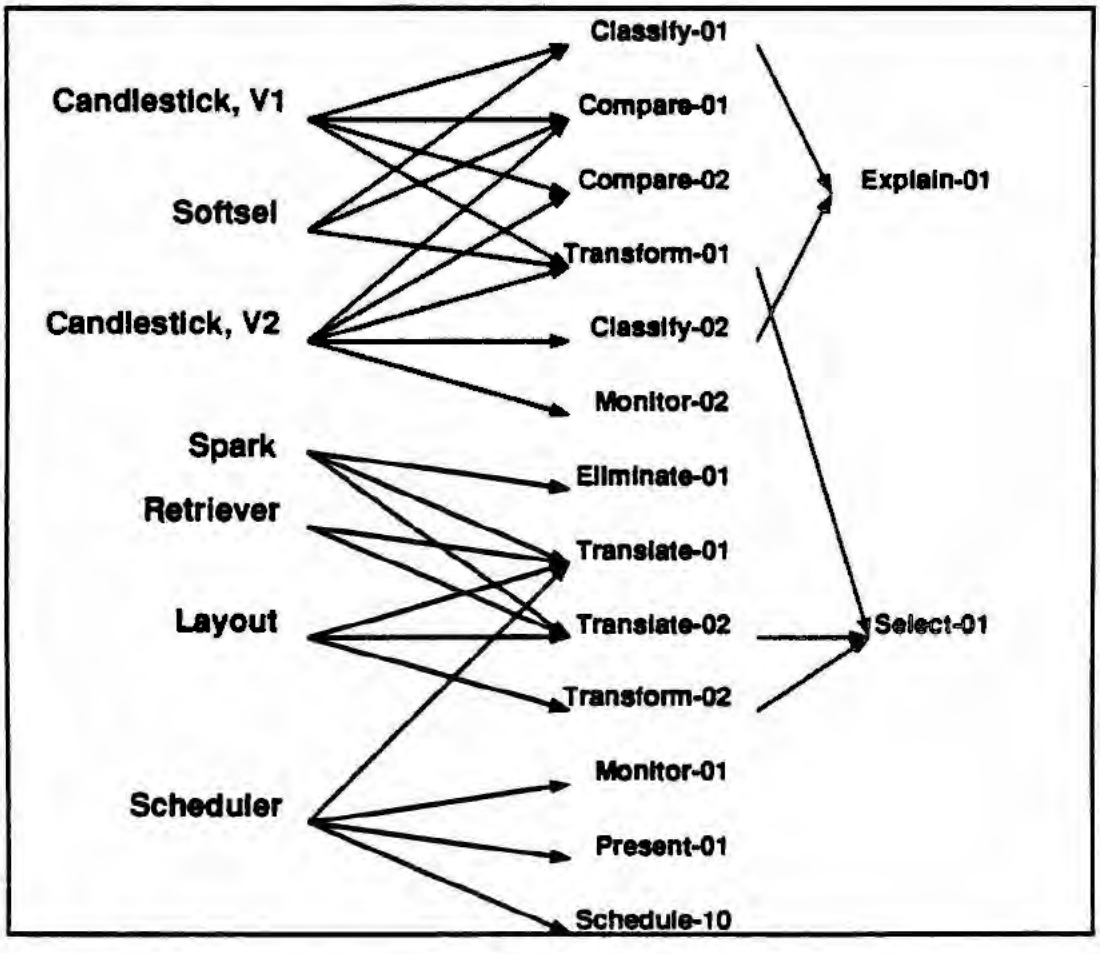


Figure 14 Mechanism Use

Figure 15 contains a brief textual description of the mechanisms.

Classify-01	The activity is to determine the classification of a set of user input based on a strong hierarchy of attributes.
Classify-02	The activity is to determine the closest classifications of a set of user input based on a very weak hierarchy of attributes.
Compare-01	The activity is to rank a set of objects for similarity with the user's description of an object based on quantitative parts of the descriptions
Compare-02	The activity is to request and compare two different solutions to a user's problem based on the two most similar stored examples.
Eliminate-01	The activity is to select a specific element from a set of possible elements based on answers to questions.
Explain-01	The activity is to show the derivation of any user selected item that was produced by adjustment of one solution to create another.
Monitor-01	The activity is to execute mechanisms based on a user's explicit preferences.
Monitor-02	The activity is to execute mechanisms in a specified order.
Present-01	The activity is to present a list on a workstation screen.
Schedule-10	The activity is to associate resources with tasks and products.
Select-01	The activity is to select a specific element from a tree, list, or set of possible elements based on the characteristics of those structures.
Transform-01	The activity is to create a solution to one problem by performing linear extrapolation on the solution to another problem, given quantitative descriptions of two problems.
Transform-02	The activity is to create a description of the spatial interdependencies for a set of rectangles.
Translate-01	The activity is to read from a file and translate its content into the data structures presupposed by mechanisms.
Translate-02	The activity is to translate the data structures presupposed by mechanisms into a pre-defined representation and write it to a file.

Figure 15 Textual Description of Mechanisms

Figure 16 presents parts of the template describing the internal representation of mechanisms.

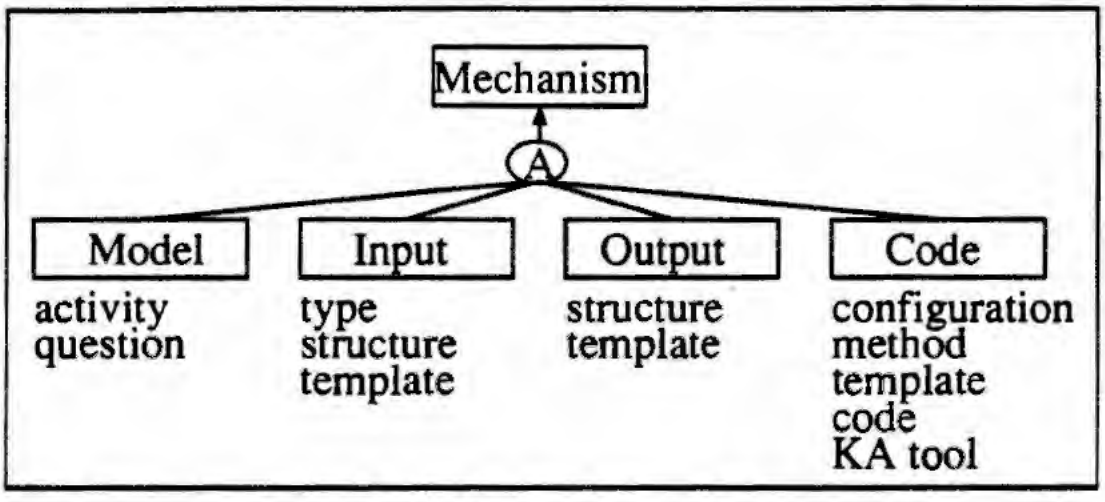


Figure 16 General Template for Describing Mechanisms

A mechanism is characterized by its

- hooks to the model: This includes pointers to the activities the mechanism implements and the questions Spark uses to differentiate among candidate configurations of mechanisms.
- input: Type information indicates whether the input is provided by the mechanism's knowledge acquisition-tool (expertise) or by another mechanism (data). Structure refers to the representation of data. Currently, lists, sets, and trees are the only data structure allowed. Each structure is composed of elements that are either primitives or list, set, and tree structures. A structure as well as an element can have attributes describing it further. The template information is a pointer to the description of the input's format. A template can be viewed as a frame whose slot values represent the actual input.
- output: The output's structure and template information is the same as for input
- hooks to code: This includes a pointer to the actual code that is not generated by the mechanism's knowledge-acquisition tool and a pointer to that tool. Configuration information consists of pointers to mechanism configurations using the mechanism. Method information describes the problem-solving method used by the mechanism. Finally, the template information is a pointer to a description of the code implementing the mechanism.

Figure 17 presents the data summarized in section 4.1.

	Configu-rations per Activity	Activities per Con-figuration
Candlestick, V1	2	1
Candlestick, V2	1	1
Layout	1	1
Retriever	1	1
Scheduler	2	1
Softsel	1	2
Spark	1	1

Figure 17 Some numbers on Mechanism Configurations

It lists the different mechanism configurations we have created so far and relates them to the activities in the enterprise model. It shows how many alternative configurations are used to implement an activity and the number of activities implemented by the same configuration.

Finally, Figure 18 presents the data summarized section 4.2. It lists the mechanisms we have

	Control Structures	Represent. per Control Structure	Levels of Detail
Classify-01	1	1	1
Classify-02	1	1	1
Compare-01	1	1	1
Compare-02	1	1	2
Eliminate-01	1	1	1
Explain-01	1	1	1
Monitor-01	1	1	1
Monitor-02	1	1	1
Present-01	1	1	1
Schedule-10	1	4	2
Select-01	3	1	1
Transform-01	1	1	2
Transform-02	1	1	1
Translate-01	3	1	1
Translate-02	2	1	1

Figure 18 Some numbers on Knowledge-Acquisition Tools

created so far and relates them to their associated knowledge-acquisition tools. It shows the number of control structures, representations, and levels of detail that are presupposed by the knowledge-acquisition tools.

This paper reported on the status of an ongoing research project. We are just at the beginning of developing a theory of mechanism. Each new application system 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.

Acknowledgments

We would be remiss if we did not mention our co-workers in this project: Renata Bushko, Michael Calvin, Glee Cameron, Rabih Chacar, Fred Conrad, Patrice Gautier, Cindy Maddern, Henrik Nordin, Charlie Reed, Tina Whitney, David Tung, and Gregg Yost are or were members of the group and made significant contributions.

References

[Bennett, 85] Bennett, J. ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. *Journal of Automated Reasoning*, 1, 1, 1985.

[Birmingham, 88] Birmingham, W. Automated Knowledge Acquisition for a Computer Hardware Synthesis System. *Proceedings of the 3rd Knowledge Acquisition for Knowledge-based Systems Workshop*. Banff, Canada, 1988.

[Boose, 84] Boose, J. Personal Construct Theory and the Transfer of Human Expertise. *Proceedings of the 4th National Conference on Artificial Intelligence*. Austin, Texas, 1984.

[Breuker 89] Breuker, J., B. Wielinga, M. van Someren, R. de Hoog, G. Schreiber, P. de Graf, B. Bredeweg, J. Wielemaker, and J. P. Billault. Model-Driven Knowledge Acquisition: Interpretation Models. *Deliverable task A1, Esprit Project 1098, Memo 87, VF Project Knowledge Acquisition in Formal Domains*, Amsterdam 1989.

[Chandra, 83] Chandrasekaran, B. Towards a Taxonomy of Problem Solving Types. *AI Magazine*, 4, 1, 1983.

[Clancey, 83] Clancey, W.J. The Epistemology of a Rule-Based Expert System -- a Framework for Explanation. *Artificial Intelligence*, 20, 3, 1983.

[Davis, 79] Davis, R. Interactive Transfer of Expertise: Acquisition of New Inference Rules. *Artificial Intelligence*, 12, 2, 1979.

[Eshelman, 88] Eshelman, L. MOLE: A Knowledge-Acquisition Tool for Cover-and-Differentiate Systems. In S. Marcus (ed), *Automating Knowledge Acquisition for Expert Systems*. Kluwer, 1988.

[Freeman, 87] Freeman, P. A Conceptual Analysis of the Draco Approach to Constructing Software Systems. *IEEE Transactions of Software Engineering*, 13, 7, 1987.

[Klinker, 88] Klinker, G. KNACK: Sample-Driven Knowledge-Acquisition Tool for Reporting Systems. In S. Marcus (ed) *Automating Knowledge Acquisition for Expert Systems*. Kluwer, 1988.

[Krueger, 89] Krueger, C. Models of Reuse in Software Engineering. Technical Report CMU-CS-89-188, Department of Computer Science, Carnegie Mellon University, 1989.

[Malone, 88] Malone T. Object Lens: a "Spreadsheet" for Cooperative Work. *ACM Transactions on Office Information Systems*, 6, 4, 1988.

[Marcus, 88] Marcus, S. SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems. In S. Marcus (ed), *Automating Knowledge Acquisition for Expert Systems*. Kluwer, 1988

[Musen, 89] Musen, M. Automated Support for Building and Extending Expert Models. *Machine Learning*, 4, 3/4, 1989.

[Newell, 81] Newell, A. The Knowledge Level. *AI Magazine*, 2, 1, 1981.

[Rich, 90] Rich, C., and R. Waters. *The Programmer's Apprentice*. Addison-Wesley Publishing Company, Inc. 1990.

[Roth, 90] Roth, S. and J. Mattis. Automatic Graphic Presentation for Production and Operations Management Systems. *Proceedings of the Fourth International Conference on Expert Systems*, Hilton Head Island, South Carolina, May, 1990.

[Yost 89] Yost, G. A Problem-Space Approach to Expert-System Specification. *Proceedings of the Eleventh International Joint Conference on AI*, Detroit, Michigan, 1989.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Theory-based Transfer of Teleological Knowledge [1]

Lawrence S. Lefkowitz[2], Dirk E. Mahling[3], and W. Bruce Croft

Collaborative Systems Laboratory
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Telephone: (413) 545-0463
Email: larry@cs.umass.edu

Abstract

The need to support the process of transferring knowledge from human experts to knowledge based systems has been recognized and extensively addressed in recent years. However, most of this work has been either ad hoc or based on theories from particular application domains. Given the level of maturity of the knowledge acquisition field, the time is appropriate to begin to formulate a uniform theory of knowledge transfer. In this paper, we address a portion of this task and propose a theoretical model as a basis for the transfer of teleological knowledge. Based on this theory, we propose a knowledge transfer architecture and show how functionality contained within existing knowledge acquisition systems can contribute to a system based on this architecture.

1 Introduction

The area of knowledge acquisition has seen a growing number of systems and ideas over the last few years (Kahn, Nowlan, and McDermott, 1985; Marcus, McDermott, and Wang, 1985; Gaines and Boose, 1988; Schirmer, 1988).

[1]This work was supported by the Office of Naval Research, under a University Initiative Research Grant, Contract # N00014-86-K-0764 and by Triumph-Adler A.G., Germany.

[2]Present address: Bell Communications Research, Knowledge-based Systems Development, RRC 1H-230, Piscataway, NJ, 08854; email: wombat@ctt.bellcore.com.

[3]Present address: DFKI, PF 2080, D-675 Kaiserslautern, Germany.

Expertise transfer systems, in particular, have been developed for many applications and purposes (Mahling and Croft, 1988; Abrett and Burstein, 1988; Boose and Bradshaw, 1988; Musen, Fagan, Coombs, and Shortliffe, 1986). These systems are mostly rooted in theories that arose from application domains. As the field matures and considerable knowledge is compiled, it seems appropriate to start the development of a unified theory of expertise transfer. Theory construction is an ongoing process in all areas of science and engineering. Increased expertise and experience lead to better theories, which in turn influence artifacts. In the scientific process, theory formation follows observation, hypothesis formation, and experiment. With the current level of observations, systems, and experience in expertise transfer we can attempt theory formation. We should be aware though, that this first attempt can only be successful if the scope of target knowledge is limited. In this paper, therefore, we restrict ourselves to teleological knowledge. Teleological knowledge is knowledge about purposeful activities, i.e., those that try to reach an objective. It is often represented in frames or plans, which consists of a number of slots for the different aspects of the teleological knowledge, such as the goal, the effects, etc.

Such a theory could integrate findings from particular domains, ideas for the specification of knowledge, observations about interactions between systems and experts, and existing theoretical approaches in related disciplines, such as psychology or educational science. An expertise transfer theory would allow one to identify the scope of existing expertise transfer systems. By matching systems with the theory, previously unexplored areas could be identified. An expertise transfer theory could be used to imply design specifications for an expertise transfer system that supports users in all stages of the transfer process.

In the following section, we present a process model as a theory of knowledge acquisition. We then review existing expertise transfer systems that contain functionality which overlaps the functionality implied by the theory (Section 3). Section 4 introduces a complete expertise transfer architecture based on the theory and utilizing functional components of the existing systems. A summary of our efforts, the status of the system, and future work are reported in Section 5.

2 A Synergistic Theory of Expertise Transfer Processes

Current attempts to model or formalize phenomena in the knowledge acquisition area are rather restricted. They either focus on knowledge transfer by a knowledge engineer (Buchanan, Barstow, et al., 1983), specific domains (Musen, Fagan, Coombs, and Shortliffe, 1986; Marcus, McDermott, and Wang, 1985), or particular psychological models (Boose, 1984; Mahling and Croft, 1988). Expertise transfer done directly by the experts can be seen as a form of end user programming. It seems therefore appropriate to start with a theory of programming and customize it to direct expertise transfer.

Pea and Kurland introduced a process model for general programming activities (Pea and Kurland, 1983). This model views programming as a cognitive activity consisting of four processes: problem comprehension, problem solution, solution coding, and code debugging. The stages can be decomposed into substages to gain further understanding. Starting with these general processes we can identify a number of subprocesses in direct expertise transfer. We extend the scope from solely human cognitive processes to joint processes of the man-machine system. Our model, shown in Figure 1, consists of *knowledge elicitation, specification, local and global verification, experimentation, and debugging.* During the elicitation process, parts of the experts' knowledge are brought to their consciousness. This conscious knowledge can then be specified by the expert. The specification process makes this knowledge available to the system. The specified knowledge can be checked for internal correctness and consistency. Inconsistencies and errors in the specified unit can be identified and corrected during the local verification and local debugging processes, respectively. The most appropriate place for the new knowledge in the existing knowledge base is found during assimilation. The global verification process checks the resulting knowledge base for semantic and syntactic correctness; the correction of any problems occurs during global debugging. Pragmatic correctness can be identified by experimenting with the new knowledge base and different test scenarios. Identification and debugging of unintended uses of the knowledge is done during the pragmatic verification and debugging processes. Note that each verification process leads to a corresponding debugging process. The system identifies problems

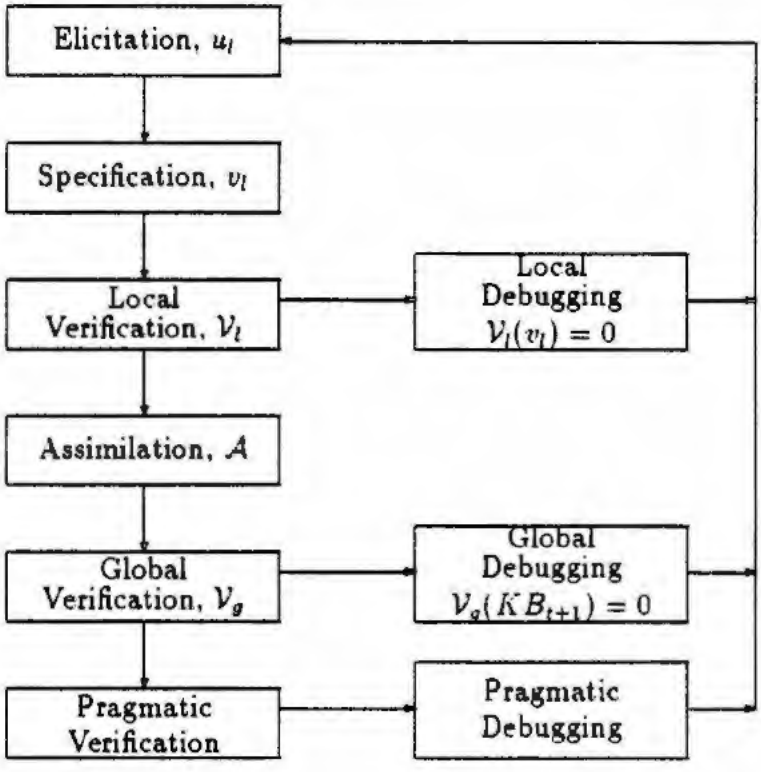


Figure 1: Expertise Transfer Processes.

in the knowledge and can start to elicit corrective information.

The first process concerns the *elicitation* of the knowledge itself. The piece of knowledge that is to be acquired by the system must be elicited from the expert. This requires at least parts of that knowledge to be present in the experts' consciousness, i.e., the experts *have to know and be aware of the answer*. To build expertise transfer systems that are usable and comfortable, the elicitation stage needs to be expanded to make predictions about the cognitive reality of the expert. The structure of the experts' memory organization, recall process, activation processes, and other cognitive entities should be regarded. Work in the area of recalling self-performed activities by Mahling and Croft (Mahling and Croft, 1989) form part of a subtheory

for this stage. Kelly's personal construct theory (Boose, 1984), Heckhausen's theory of motivation (Heckhausen and Kuhl, 1985), and Hacker's engineering psychology (Hacker, 1985) can also support this first stage.

We can formalize this stage of the expertise transfer process using Ontological Analysis (Alexander, Freiling, Shulman, and Stanley, 1986) and techniques applied in Mathematical Psychology (Fisher, 1988). Ontological Analysis helps identify and formalize core concepts of a task under study while Mathematical Psychology helps us to make psychological phenomena tractable in a formal way. We can move from the ontology, which tries to capture the reality of a domain, to epistemology, the person's representation of that domain. Let E_i be the expertise of expert i, SE_i the static epistemology (facts and properties of a domain) and DE_i the dynamic epistemology (entities that bring about change in a domain). We can then say the following:

$$E_i = DE_i \cup SE_i \tag{1}$$

meaning that this expert's knowledge is comprised of the three ontologies. We can further say that the expertise in an area is comprised by the knowledge of the experts in that area:

$$E = \bigcup_{i=1}^{n} E_i. \tag{2}$$

As a first approximation, we can formalize the results of the elicitation process to:

$$u_l \approx \sigma(E_i) \tag{3}$$

where σ is a selection, or awareness function, and u_l is a unit of knowledge (Date, 1981; Holland, Holyoak, Nisbett, and Thagard, 1986).

Motivating an expert to transfer knowledge is the key to successful knowledge acquisition. Experts rarely have the time or the willingness to cooperate and make major commitments when the system has no immediate relevance to them. Fears of being replaced by the system, regardless of how unrealistic they might be, might lower motivation even further. Recognizing motivation as a prominent factor for correct elicitation, we can refine equation 3 to:

$$u_l = m_r(\sigma E_i) \tag{4}$$

where m_r reflects the experts motivation to recall. We can view m_r as a multiplying factor in the range $[0, 1]$. Heckhausen's (Heckhausen and Kuhl, 1985) motivation model provides a possible basis for this aspect of the theory.

Consider, for example, the domain of conference planning. To acquire the knowledge of how a symposium is staged, this knowledge (DE_i) has to be present in the mind of the expert. The expert has to understand that the general task of symposium planning is involved. The expert recalls the different ways symposia can be organized, the problems associated with it, the tasks from the dynamic epistemology DE_i and people from the static epistemology SE_i involved, previous incidents, etc. Coherent chunks of expertise in the person's knowledge structure are activated (u_l). This knowledge is ready to be transferred.

The elicited piece of knowledge, u_l, must be transferred by the experts from their consciousness to the system. This stage is called *knowledge specification*. The entities that comprise the elicited knowledge must be named and their relations must be defined. We can view this as a mapping function from the conscious knowledge of the expert into the system's representation:

$$v_l = m_s S(m_r \sigma(E_i)) \tag{5}$$

or after substitution

$$v_l = m_s S(u_l) \tag{6}$$

where v_l is a unit in the knowledge base of the expert system, m_s is the expert's motivation to specify the recalled knowledge, and S is the experts' mapping from their own knowledge to the system's representation. S can vary according to the individual expert.

To capture the experts' knowledge correctly, S has to preserve the experts' knowledge structure. Units that are related to each other in E_i must be related in KB. This property of S is known as homomorphism. To prove that S is homomorphic for a given transfer language, a given epistemology, and a given knowledge representation formalism, we have to show that $\forall u, v s.t u \in E \wedge v \in KB$ the function S fulfills:

$$S(u_l) + S(u_m) = S(u_l + u_m) \quad \text{and} \tag{7}$$

$$S(cu_l) = cS(u_l) \tag{8}$$

S is preserving the structure of the experts knowledge and allows specification of all units of knowledge. Units that are interrelated in the experts epistemology, must also be interrelated in the knowledge base (equation 7). Performing a cognitive operation on a unit, such as scaling or updating, must also be possible in the knowledge base (equation 8). It might not be possible to show that the inverse of S is also homomorphic. This would imply an isomorphism between the experts epistemology and the knowledge base, meaning that the knowledge base is an identical copy of the experts knowledge. If this would be true, psychology would be obsolete.

The expertise transfer system must provide a facility to support the experts' mapping, S, while guaranteeing a high motivation, m_s. After the specification stage is completed, the elicited knowledge is transferred to the system. A first approximation of the knowledge is represented in a formal way in the underlying formalism of the system, v_l. It is not necessary to acquire a complete description of a knowledge unit in terms of the underlying formalism. Parts of the unit can still be unspecified. Knowledge can be acquired from multiple experts or during consecutive sessions. Incomplete units can be incrementally updated.

In our example, the experts would have to use an expertise transfer language to specify their knowledge that staging a symposium consists of three high level steps: preparing the symposium, conducting it, and writing a final report. This specification process can result, depending on the implementation of the expertise transfer language, in a collection of terms like:

```
(Stage.A.Symposium = (first (Prepare.Symposium)
                     then  (Conduct.Symposium)
                     last  (Write.Final.Report)))
```

The motivation m_s for actually entering the recalled knowledge is as crucial as the general willingness of the expert to cooperate. Units should not be too large, as people strive for psychological closure. The theorem

$$v_i \prec v_j \Rightarrow m_{s_i} \succ m_{s_j} \tag{9}$$

follows from this observation, where $\prec$ indicates a preference or psychological magnitude relation, as used by Coombs, Dawes, and Tversky (Coombs,

Dawes, and Tversky, 1970). Immediately rewarding users for entered knowledge enhances m, and helps to sustain the expertise transfer process.

Locally verifying the semantic and syntactic consistency of the new knowledge unit is done next. The new unit v_l is regarded in isolation from the knowledge base. Internal semantic inconsistencies and ambiguities can be detected. Missing parts and violations of the representational formalism can be found by a syntactic analysis. We can define a verification function $\mathcal{V}_l$, which returns 1 if and only if the local unit is consistent:

$$\mathcal{V}_l(v_l) = \begin{cases} 1 & \text{if } v_l \text{ is consistent} \\ 0 & \text{otherwise.} \end{cases}$$

If $\mathcal{V}_l$ returns 0, ambiguities or other violations are found, the expert can be asked to correct them. The sources for the problems are known and users can immediately be directed to the trouble spots. Users can locally debug the unit, where the problems serve as cues for correct elicitation. This is an iterative process, terminating with the completion of a locally correct new unit v_l.

The locally correct new unit v_l has to be added to the knowledge base KB. This process is known as *knowledge assimilation*. The best ties to existing knowledge within the existing knowledge base have to be found. For a given knowledge base KB and a new piece of knowledge v_l, this process can be formalized as

$$KB_{t+1} = \mathcal{A}(KB_t, v_l) \tag{10}$$

where $\mathcal{A}$ is an assimilation function and KB_t is the knowledge base at time t.

Adding activity knowledge, such as WRITE.FINAL.REPORT, could be done by searching for the closest matching unit. This unit could be WRITE.DOCUMENT, and WRITE.FINAL.REPORT could be made a subclass of the more general activity WRITE.DOCUMENT.

Contradictions and other sources of incongruency with existing knowledge are detected. during the *global verification* process. Using the verification

function $\mathcal{V}_g$, we can extend its definition to:

$$\mathcal{V}_g(KB_{t+1}) = \begin{cases} 1 & \text{if } \mathcal{A}(KB_t, v_l) \text{ is consistent} \\ 0 & \text{otherwise.} \end{cases}$$

If no inconsistencies are found, the new unit is tentatively added to the knowledge base. As we will see, the final decision to enter it depends on the results of the experimentation process.

The *global debugging* processs is only necessary if $\mathcal{V}_g(KB_{t+1}) = 0$. In that case, inconsistencies must be resolved either by querying the user or by automatic means.

A *pragmtaic verification* process is needed to guarantee pragmatic consistency in the updated knowledge base KB_{t+1}. The behavior of the updated knowledge base, employing the tentatively added piece of knowledge, must be presented to the domain expert. This enables the expert to recognize dynamic relations between existing and new pieces of knowledge and to verify that these interactions were intended.

Pragmatic inconsistencies could arise in our example, if the a unit v_l from the dynamic epistemology DE, such as PREPARE.SYMPOSIUM used the object CONFERENCE.HOTEL and CONFERENCE.ROOM. The object, CONFERENCE.HOTEL for instance, might already exist in the knowledge base. If CONFERENCE.ROOM is not constrained to CONFERENCE.HOTEL, we might end up with the attendees living in one hotel, while the conference takes place in another one. Pragmatic verification should help users detect such cases.

Showing this unintended interaction between syntactically correct units can only be achieved by exercising the knowledge base. As it is extremely hard to automate the verification of epistemological knowledge, which requires a pragmatic understanding of the domain, experts are encouraged to experiment with the new knowledge base. Deploying test scenarios to evaluate the knowledge base in a number of different situations can enable the experts to identify knowledge that is over- or under-constrained. Experimentation can also help to reveal unintended knowledge base interactions.

If undesired semantic and pragmatic interactions between pieces of knowl-

edge were discovered, it is necessary to respecify the tentatively added piece of knowledge. Like local debugging and global debugging, this process is essentially a debugging activity. Problems are isolated as they appear. Additional elicitation and specification can occur in the context of experimentation.

3 Drawing on Existing Research

A number of systems in the area of expertise transfer have functionality that can immediately be matched to subprocesses in the expertise transfer theory. This section reviews some of the major expertise transfer system to identify such functionalities. SALT (Marcus, McDermott, and Wang, 1985) and DACRON (Mahling, 1990) are particularly interesting for their user interface and elicitation techniques. The K^n_{AC} (Lefkowitz and Lesser, 1988) system carries the process on by assimilating knowledge and generating expectations for new knowledge, thereby pertaining to assimilation and global verification. Most knowledge acquisition system are embedded in an expert system which uses the acquired knowledge to give advice or perform intelligent actions. Such functionality can help in the pragmatic verification of acquired knowledge. DACRON is an example for an acquisition/advice system that allows modification during experimentation, thereby integrating pragmatic verification and pragmatic debugging.

The SALT system guides interviews with a domain expert to perform knowledge acquisition for configuration tasks. It uses knowledge about the specific structure types (e.g., *method*, *constraint* or *fix*) and the already existing configuration knowledge to check and restrict the information entered by the expert. SALT relates mostly to the elicitation process and the resulting unit u_l. The gaps in the SALT's knowledge base serve as expectation for further knowledge acquisition and can be seen as elaborate debugging processes in our theory.

DACRON allows users to specify elicited knowledge graphically and verifies the new units locally. The modules in DACRON provide a basis for S, the specification subprocess, and $\mathcal{V}_l$, the local verification subprocesses, while keeping the m_r and m_s, the recall and specification motivation, at high levels. The graphical editors in DACRON could also be used to present the planning

process of an AI system during the pragmatic verification process. Experiments with the system have shown that experts could identify pragmatic problems in this stage by looking at graphical animations of the planning and variable binding behavior of the system. Graphical editors used in the elicitation and specification stage have proven useful as devices for immediate debugging of pragmatic errors.

The K^n_Ac system was developed to assist in the assimilation $\mathcal{A}$ of domain knowledge presented by a domain expert into an existing knowledge base. To help focus this process, the K^n_Ac system would anticipate modifications to the knowledge base and use the expectations to focus in on small areas of the existing knowledge KB_t. New information v_l from the domain expert was compared to existing "in focus" entities to obtain similarities and differences. The match was evaluated both in terms of how similar the entities were as well as how closely the modifications implied by the differences matched the anticipated knowledge base changes. K^n_Ac used a variety of heuristics to generate its modification expectations, including heuristics based on the state (completeness and consistency) of the existing knowledge base, guidance provided via the acquisition discourse, and heuristics based on the knowledge acquisition strategy being employed.

Whereas K^n_Ac was designed specifically to aid in the knowledge acquisition process, the SPANDEX system (Broverman and Croft, 1988) was created to allow recovery from exceptional behavior during the execution of a plan generated from an incomplete (or incorrect) domain model. This functionality pertains to the pragmatic verification of the knowledge base. Because the agents performing application tasks may have knowledge beyond that in the planner's domain model, SPANDEX allows for the possibility that an agent's behavior which is not anticipated by the planner is not simply erroneous. Rather, SPANDEX attempts to explain the behavior in terms of the current task context. Some explanations may require modifications to the domain model which provides an opportunity for expertise transfer. Thus, differences between the planner's expectations and the user's behavior allow SPANDEX to identify and correct potential problems with the existing knowledge base.

4 A Generic Architecture to Support all Stages of Knowledge Acquisition

Each of the processes and systems described in the previous sections provide an important functionality needed for the successful transfer of expertise. However, no single approach is sufficient. To help a user modify or augment the teleological knowledge contained in an existing domain model, a teleological expertise transfer facility must be able to present existing knowledge to the user, ask the user for additional information, validate the information it is given, assimilate the new information with what is already known and allow the user to experiment with the newly assimilated knowledge. Figure 2 shows how the process model of expertise transfer relates to modules in existing expertise transfer systems, and how these in turn can be integrated in a combined architecture. In the remainder of this section, we describe that architecture (Figure 3) in more detail.

4.1 Presentation and Elicitation

To insulate the typical end-user or domain expert from the internal knowledge representation, teleological knowledge must be presented to and elicited from the user in an intuitive and aesthetically acceptable format. As in a User-Interface Management System (UIMS), the Presentation and Elicitation component can use presentation knowledge, dialog knowledge and user models to control *how* information is presented or elicited. This helps to engineer an interface that approximates the experts' mapping, $\mathcal{S}$, very closely. As part of an integrated acquisition system, context derived from the expertise transfer process can also be used to refine the manner in which information is presented or elicited (Mahling and Lefkowitz, 1989).

In addition to controlling the style of interaction, this architecture enables feedback from the other modules to be used to determine *what* information to present or elicit at a given time. Specifically, shortcomings detected by the Validation $\mathcal{V}$, Assimilation $\mathcal{A}$, and Experimentation modules can advise the Elicitation component as to what additional information is required. Furthermore, by understanding the need for the requested knowledge, the

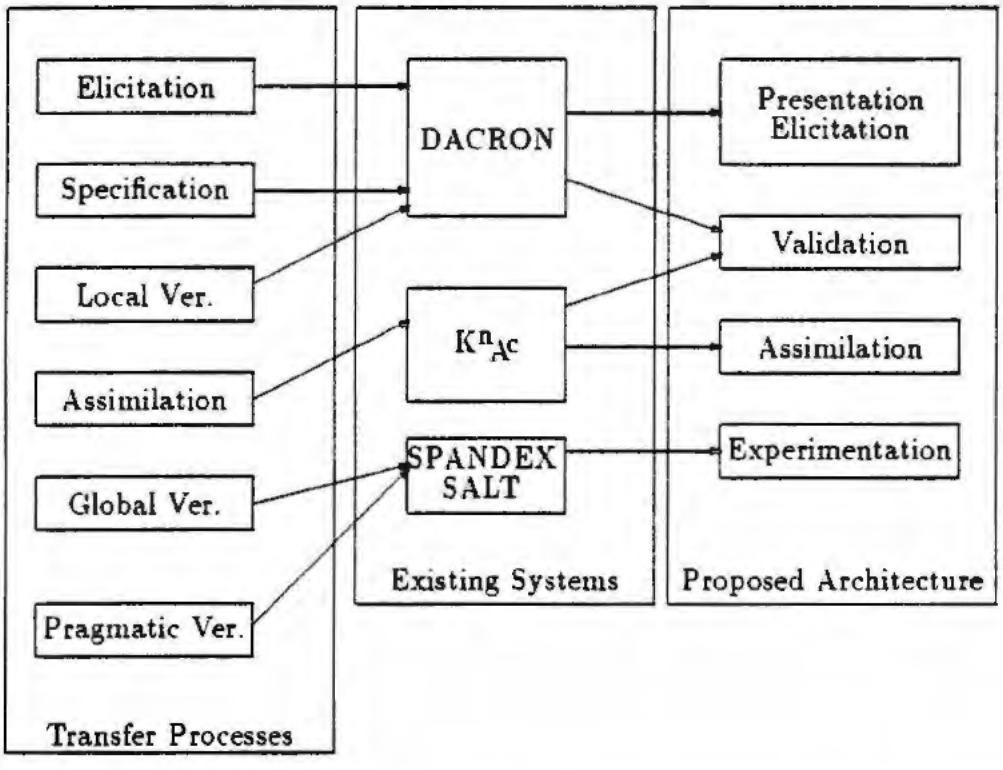


Figure 2: Relations between the Theory, Existing Systems, and the Proposed Architecture.

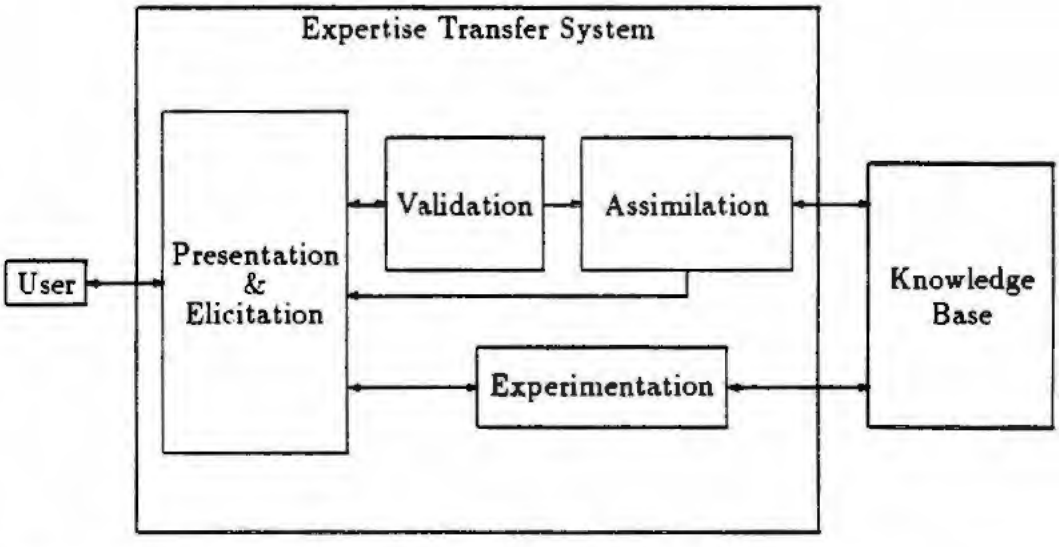


Figure 3: An Architecture for Acquisition of Teleological Knowledge

system is better able to select which information to display to provide an appropriate context, mirroring $\sigma(E_i)$, in which to query the user. The question whether the relation between E_i and KB is homomorphic or isomorphic is of immediate importance for this module.

4.2 Local Validation

The more elaborate knowledge representation used in representation of teleological knowledge allows an associated level of semantics which is not found with simpler representations. Thus, in addition to checking the syntactic validity of information obtained by the Presentation and Elicitation module, the Local Validation component can check the new entities for "completeness" and "consistency." These can involve heuristic tests, some which are domain-independent and others which require application-specific knowledge. Meta-information about the entities (e.g., datatype, cardinality, etc.) provide one source for the metrics; domain constraints provide another. Inconsistencies or incompleteness detected by this validation process are used to guide the elicitation of additional information. The function $\mathcal{V}$ can be tailored to the domain and the underlying knowledge representation.

4.3 Assimilation and Global Verification

A much touted feature of production rule systems is the independence of individual rules. While strictly true, the functionality of the resulting knowledge base is very dependent on the interactions among the individual chunks of knowledge $v_1, v_2, ..., v_n$. The more structured teleological knowledge enables more of these relationships to be made explicit while maintaining a high degree of modularity. Thus, new information should not simply be appended to the existing knowledge base; rather, it should be assimilated into existing knowledge structures using the function $\mathcal{A}$. By so doing, conflicts which are not completely local to the new information but rather arise from its interaction with other entities can be detected using $\mathcal{V}$ in the global verification process. As with the local validation model, these conflicts can guide the acquisition process to modify either the new unit v_l or the relevant parts of the knowledge base KB_t.

Let us consider a particular assimilation function $\mathcal{A}$: A first step in assimilating new information v_l is to compare it to portions of the existing knowledge base KB_t. As the size of knowledge base KB grows, the need to focus the matching process on selected areas becomes more important. By anticipating modifications to the existing knowledge base, the acquisition system can focus the matching process on those entities $v_x, v_y, ..., v_z$ where change is expected. Furthermore, the anticipated changes to the existing structures can be used in the evaluation of the matches; differences between the new and existing knowledge which fulfill the expectations add, rather than detract, from the quality of a match.

4.4 Testing and Experimentation

Teleological knowledge is provided with the expectation that it will contribute to the formation of a plan to achieve some goal in the application domain. The most reasonable way of allowing the user evaluate the effects of the newly added or modified knowledge v_l is to "exercise" the modified portions of the knowledge base KB. Thus, if descriptions of domain activities or actions in DE have been added, scenarios should be generated that construct plans that use the new activities. Object descriptions emanating from SE can

be tested by creating plans which should apply to those objects. Similarly, confidence in the descriptions of new agents can be increased by invoking activities that require those agents. Thus, after attempting local and global validation $\mathcal{V}$ of new information v_i, a final test is provided experiment with ways in which the new information actually performs.

The inability to generate a plan or unexpected performance resulting from newly added knowledge offers insights into necessary knowledge base modifications. By allowing the user to identify portions of the system generated plans which do not conform to the user's expectations, the acquisition system can generate explanations of how the current domain model can be adapted to conform with the user's expectations. These explanations provide a set of potential modifications to the existing knowledge base which can be used to guide the acquisition process.

4.5 Feedback

As mentioned in Section 4.1, the Presentation and Elicitation module determines how information is presented to or elicited from the user. The feedback provided from the other modules provides guidance as to what additional or modified information needs to be obtained and what existing information should be shown. The feedback provided by each module was described briefly in each of the preceding sections; it can be summarized as follows:

- Local validation identifies inconsistencies or incompleteness within a given chunk of knowledge. The missing portions of the new information as well as the violated constraints (or meta-knowledge) are identified as candidates for modification.
- Assimilation recognizes more global problems resulting from the addition of the new knowledge. This process highlights other entities affected by the new knowledge which may need to be modified.
- Experimentation with the newly provided knowledge can lead to erroneous or unexpected system performance. Analysis and explanations of the errors or exceptions can yield suggestions to guide the knowledge modification.

5 Summary and Future Work

The main objective of the Collaborative Systems Laboratory's POLYMER project has been to support the planning and execution of tasks in real world domains. As such, the representation, acquisition and presentation of teleological knowledge has been a central issue within the project from its inception and has been partially addressed by several research programs within the lab. We feel the time is appropriate to extract from our varied experiences within the POLYMER project, as well as from related research in the fields of knowledge acquisition, interface design and cognitive psychology, an integrated theory describing the transfer of teleological knowledge from human experts to a knowledge-based system. The formation of such a theory is an integral component in the development of robust and cognitively sound expertise transfer systems.

Based on this integrated theory, we have proposed an architecture for a knowledge transfer system and, where possible, have related the functional modules within this architecture to existing software systems. Our immediate goal is to refine this architecture and implement an experimental knowledge transfer system.

Our purpose in constructing such a testbed is twofold. First, while evaluation has been performed on some system components, most notably the DACRON front-end and the SPANDEX exception handler, other capabilities such as knowledge assimilation and global verification have been more difficult to evaluate without an appropriate user interface. By developing an integrated transfer system, we can better evaluate the performance of each of the separate components. The second, and perhaps more difficult issue, is the understanding of the interactions among the functional modules. By evaluating the success of the system as a whole while controlling the contribution of each component, we hope to gain an understanding of the appropriate role of each module and required interactions. Previous success in usability testing of the DACRON interface style combined with POLYMER functionality implies that such a user-centered approach to the development of expertise transfer systems is both desirable and feasible.

References

Abrett, H. and Burstein, M.H. (1988). The KREME Knowledge Editing Environment, in J. Boose and B. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*, Academic Press.

Alexander, J.H., Freiling, M.J., Shulman, S., and Stanley, H. (1986). Knowledge Level Engineering: Ontological Analysis, in *1986 National Conference on Artificial Intelligence*, AAAI.

Boose, J. (1984). Personal Construct Theory and the Transfer of Human Expertise. in *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas.

Boose, J.H. and Bradshaw, J.M. (1988). Expertise Transer and Complexc Problems: Using AQUINAS as a Knowledge-acquisition Workbench for Knowledge-based Systems, in J. Boose and B. Gaines (Eds.), *Knowledge Acquisition Tools for Expert Systems*, Academic Press.

Broverman, C.A. and Croft, W.B. (1988). Plausible Explanation to Cope with Unanticipated Behavior in Planning, Technical Report 88-56, University of Massachusetts.

Buchanan, B.G., Barstow, D., Bechtal, R., Benett, J., Clancey, W., Kulikowksi, C., Mitchell, T., and Waterman, D.A. (1983). Constructing an Expert System, in F. Hayes-Roth, D. A. Waterman, and D. B. Lenat (Eds.), *Building Expert Systems*, pages 126 –167, Addison-Wesley, Reading, Massachusetts.

Coombs, C.H., Dawes, R.M., and Tversky, A. (1970). *Mathematical Psychology*, Prentice Hall.

Date, C. (1981). *An Introduction to Database Systems*, Addison Wesley.

Fisher, D.L. (1988). Optimization in Cognitive Engineering, Lecture Notes for Cognitive Engineering Seminar, IEOR 689, Department of Industrial Engineering, University of Massachusetts.

Gaines, B.R. and Boose, J.H. (1988). Knowledge Acquisition Tools for Expert Systems, in B.R. Gaines and J.H. Boose (Eds.), *Knowledge Acquisition Tools for Expert Systems*, pages xiii–xvi, Academic Press.

Hacker, W. (1985). Activity: A Fruitful Concept in Industrial Psychology, in J. Kuhl and J. Beckmann (Eds.), *Action Control: From Cognition to Behavior*, Springer-Verlag.

Heckhausen, H. and Kuhl, J. (1985). From Wishes to Action: The Dead Ends and Short Cuts on the Long Way to Action, in J. Kuhl and J. Beckmann (Eds.), *Action Control: From Cognition to Behavior*, Springer-Verlag.

Holland, J.H., Holyoak,K.J., Nisbett, R.E., and Thagard, P.R. (1986). *Induction: Processes of Inference, Learning and Discovery*, MIT Press.

Kahn, G., Nowlan, S., and McDermott, J. (1985). MORE: An Intelligent Knowledge Acquisition Tool, in *Proceedings of the Ninth International Conference on Artificial Intelligence*, Los Angeles, California.

Lefkowitz, L.S. and Lesser, V.R. (1988). Knowledge Acquisition as Knowledge Assimilation, in *International Journal of Man-Machine Studies*, 29(2):215–226. (Also in Machine Learning and Uncertain Reasoning, Academic Press Limited, 1989.)

Mahling, D.E. (1990). *A Visual Language for Knowledge Acquisition, Display and Animation by Domain Experts and Novices*, PhD thesis, University of Massachusetts.

Mahling, D.E. and Croft, W.B. (1988). Knowledge Acquisition for Planners, in B. Gaines and J. Boose (Eds.), *Proceedings of the 3rd Knowledge Acquisition for Knowledge Based Systems Workshop*.

Mahling, D.E. and Croft, W.B. (1989). Relating Human Knowledge to Plans, in *International Journal of Man-Machine Studies*, 31:61 – 97.

Mahling, D.E. and Lefkowitz, L.S. (1989). Using Task Context to Support an Adaptable Interface, in G. Salvendy, editor, *Designing and Using Human-*

Computer Interfaces and Knowledge Based Systems, pages 565–573, Elsevier, Amsterdam.

Marcus, S.J., McDermott, J., and Wang, T. (1985). Knowledge Acquistion for Constructive Systems, in *Proceedings of the Ninth International Conference on Artificial Intelligence*, Los Angeles, California.

Musen, M., Fagan, L., Coombs, D., and Shortliffe, E. (1986). Using a Domain Model to Drive an Interactive Knowledge Editing Tool, in *Proceedings of the Knowledge Acquisition for Knowledge Based Systems Workshop*, pages 33.0–33.11, American Association for Artificial Intelligence.

Pea, R.D. and Kurland, D.M. (1983). On the Cognitive Prerequisites of Computer Programming, Technical Report TR 18, Bank Street College, New York.

Schirmer, K. (1988). Techniken der Wissensakquisition. *Kuenstliche Intelligenz: Forschung, Entwicklung, Erfahrungen*, 4/88:68 –71.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Hyper-KSE: A Heterogeneous Integration of Acquisition, Interpretation and Documentation Tools

Marc Linster[1] & Brian Gaines[2]

[1] Expert Systems Research Group
GMD
PO Box 1240
D-5205 St. Augustin 1, FRG
e-mail: linster@gmdzi.gmd.de

[2] Knowledge Science Institute
University of Calgary
2500 University Drive, NW
Calgary, Canada T2N 1N4
e-mail: gaines@cpsc.ucalgary.ca

We motivate and present a loosely coupled environment (Hyper-KSE) that integrates a knowledge acquisition tool (KSS0) with an inference tool (BABYLON) and a hypermedia system (HyperCard). KSS0 is used to acquire the knowledge which is then automatically cross-compiled into a BABYLON knowledge-base. HyperCard allows for an on-line documentation of the knowledge that is introduced in the system during the acquisition process, and for the use of this documentation while the performance system is querying the user during a consultation.

After a brief introduction of the principles of the translation of KSS0 knowledge bases into BABYLON knowledge bases a simple example is used to illustrate the interaction of the tools in the development of a knowledge base, and to explain why such an integration of tools using different media opens qualitatively new dimensions for knowledge based systems.

1. MOTIVATION

Most current knowledge acquisition and interpretation environments, e.g. OPAL, PROTÉGÉ (Musen, 1989) or CLASSIKA (Gappa, 1989)

- rely on symbols only when communicating with their users, such that the users of the knowledge-based system must have a deep understanding of the domain, just to be sure that their conceptual models coincide with the conceptual model of the developer of the knowledge based system;

- knowledge use and knowledge acquisition happen in two different environments, so that incremental development, modification, and maintenance is quite difficult while the system is being used;

- the analysis tools used during the acquisition phases are out of the user's reach if he/she wants to get a deeper understanding of the knowledge base.

Hyper-KSE (Hypermedia-based Knowledge Support Environment) is an integration of three tools: KSS0 elicits, acquires and structures knowledge; HyperCard provides a link between knowledge outside of knowledge-based systems (e.g. in books, graphics, pictures or motion-sequences); BABYLON interprets the knowledge to solve problems during a consultation.

In this paper we want to show how the above mentioned problems can be tackled through the integration. The technical aspects of the integration of the tools are discussed in (Gaines and Linster, 1990).

2. THE TOOLS

We will give brief overviews of KSS0 and BABYLON to show the features of the tools as far as they are relevant for the discussion in this paper. For detailed discussions see (Gaines, 1989) and (Christaller, DiPrimio and Voß, 1989).

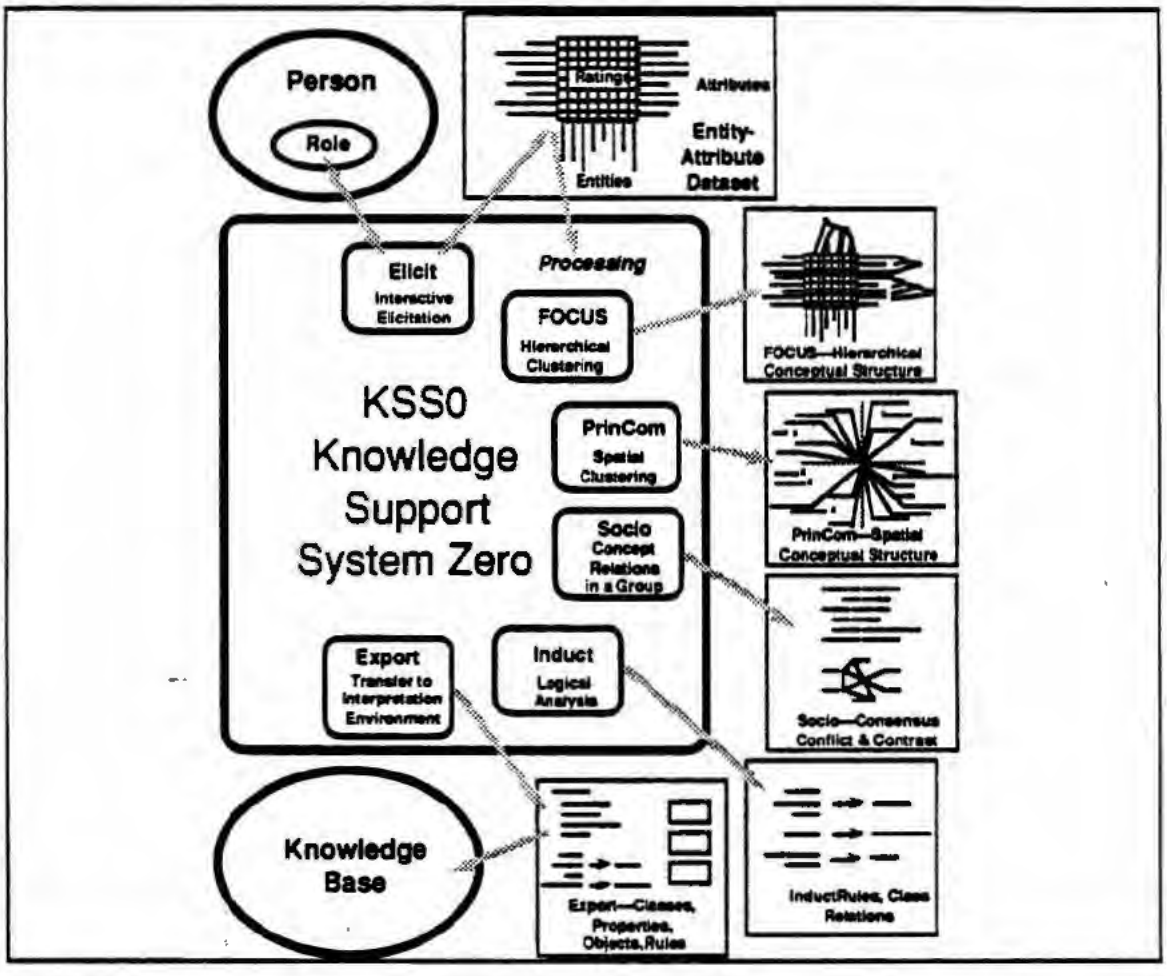


Fig. 1: The architecture of KSS0

2.1. KSS0

KSS0 (Knowledge Support System Zero) is a knowledge elicitation, acquisition and structuring tool that combines repertory-grid techniques (Shaw and Gaines, 1987, Kelly, 1955) with visual feedback and analysis techniques to acquire knowledge and to help the user find hidden structures and at the same time get new insights about his own knowledge. As "seeing the knowledge in action" provides important feedback to extend the knowledge base, KSS0 can export its knowledge base to environments that can interpret this knowledge during a consultation (see fig. 1). KSS0 exports its knowledge base in form of a set of examples that are described by attributes with values. For every attribute it indicates the weight (a numeric rating to describe priorities) and the list of possible values. A list of rules describes regularities in the use of the attributes and values in the description of the attributes.

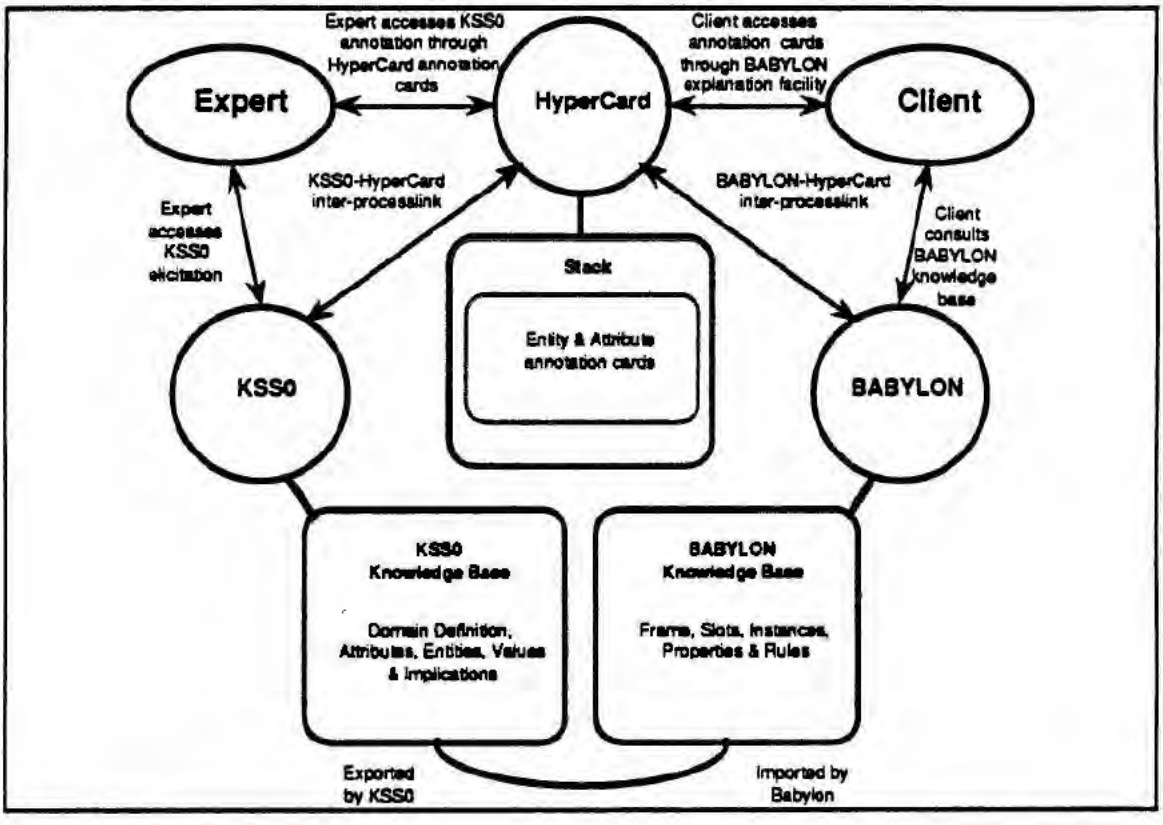


Fig. 2: The interaction of the tools within Hyper-KSE

2.2. BABYLON

BABYLON (DiPrimio and Wittur, 1987, Christaller, DiPrimio and Voß, 1989) is a hybrid knowledge representation and interpretation environment that combines rules, clauses, frames and constraints in a meta-processor architecture such that all these formalisms can interact tightly within the same knowledge-base. BABYLON is a highly flexible environment that does not have any built-in problem-solving methods, as e.g. MED-2 has a built-in differential diagnosis mechanism (Puppe, 1989) or ONCOCIN includes a form of skeletal-plan refinement

(Tu et al, 1989). Thus BABYLON can be used for any problem-solving method, but the method must be hand-coded by the knowledge engineer.

3. THE SCHEME OF THE INTEGRATION

KSS0 is used to elicit knowledge from an expert and to work it into structures. The data elements of a KSS0 knowledge base are annotated using a special HyperCard stack (see fig, 3) that contains one annotation card for each data element. These cards contain the list of possible-values for attributes, and the list of attribute-value pairs for concepts described by attributes. Additionally free text comments can be entered. The cards can be handles for arbitrary suits of other cards that can be linked using standard HyperCard facilities.

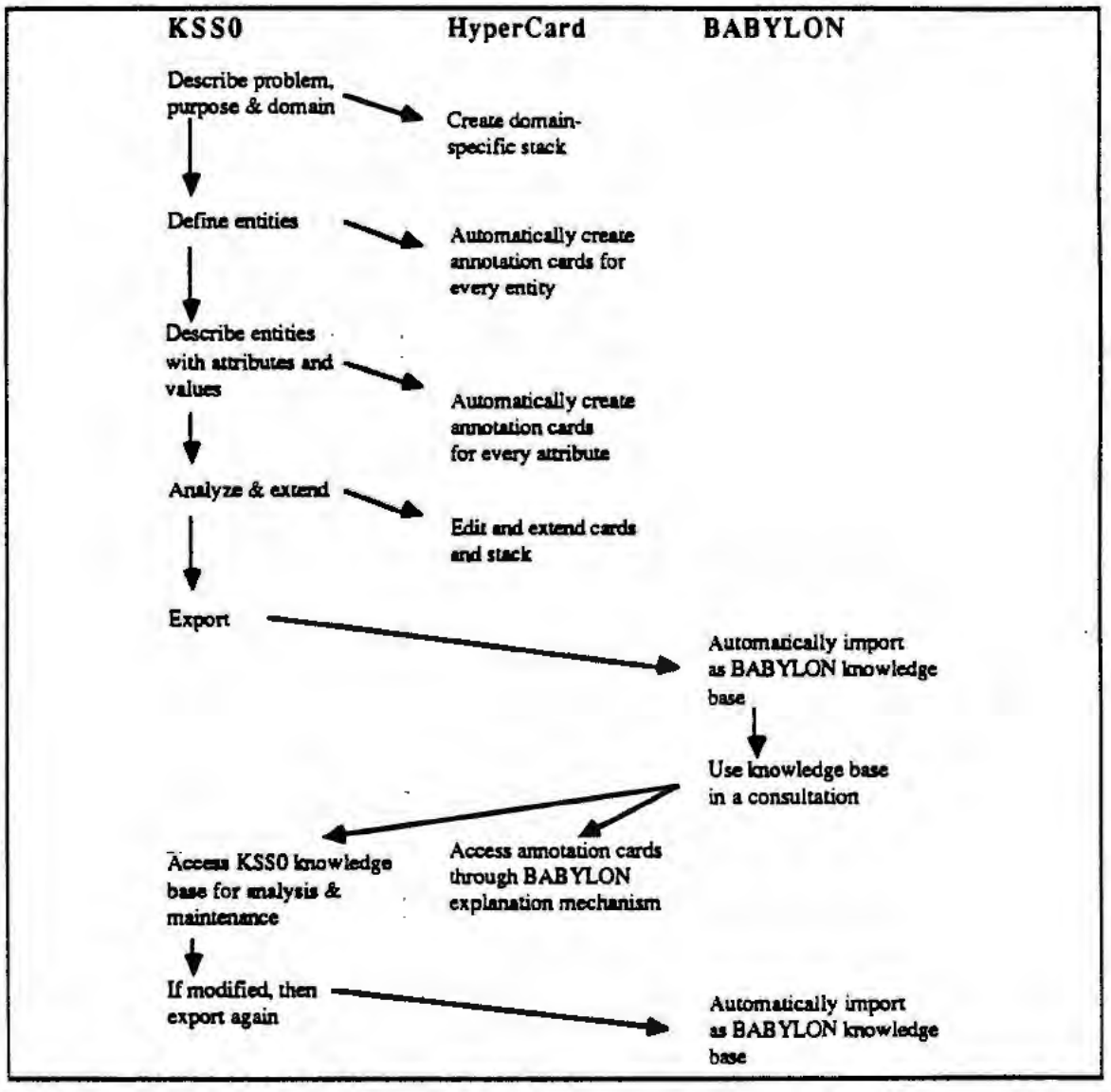


Fig. 3: The sequence of the use of the tools when developing, consulting and maintaining a knowledge base with Hyper-KSE.

Using KSS0's export mechanism a BABYLON knowledge base is written (see fig. 3). This can then be used during a consultation. The annotation cards can be accessed during the consultation to see what the expert meant when using an attribute, a value or what the rationale for a rule is. At the same time the client can access the KSS0 knowledge base to inspect it or introduce new knowledge elements if the result of a consultation do not meet his expectations (see fig. 2 & 3).

In case the KSS0 knowledge base has been modified, it must be exported again to keep BABYLON informed.

4. THE INFERENCE PROCESS

The KSS0 knowledge base exported to BABYLON contains among others the definitions of attributes and implications between these attributes derived by INDUCT (Gaines, 1989), e.g.:

```
(Ind_4
 (?And (Mushroom Underside_Of_Cap = Lamella/Gill)
       (Mushroom Surface = Smooth))
 ($Execute (Mushroom Edibility = Edible)))

(Ind_9
 (?And (Mushroom Underside_Of_Cap = Tubes)
       (Mushroom Stalk = Thin))
 ($Execute (Mushroom Edibility = Poisonous)))
```

Fig. 4: Two of the rules used to infer the value of the attribute *edibility*.

The rule Ind_4 (fig. 4) expresses that, given the examples and the use of the attributes in the KSS0 knowledge base, it is reasonable to infer that if the underside of the cap of a mushroom is made out of tubes and its stalk is thin, then it is a poisonous mushroom.

This knowledge is used to implement a heuristic association process (Karbach and Linster, 1990) in BABYLON, i.e. from values of attributes BABYLON concludes values of other attributes. KSS0 distinguishes input attributes and output attributes. The values of input attributes are queried (if needed for an inference) from the user, the values of the output attributes are inferred using the implication rules and the values of the input attributes. As an attribute can be input and output attribute at the same time, and as priorities among the attributes are used to define which output attribute to infer next and which input attribute to inquire about first, quite complex reasoning processes can be implemented.

In the operational knowledge bases of Hyper-KSE this means that there are rules that infer values of attributes (see fig. 4), and there are control rules (see fig 5) to decide which output-attribute to infer next:

```
(Edibility
    ($And (Mushroom Edibility = -))
    ($Execute (Find-Implications :Edibility-Rules :Do-One))))
```

Fig. 5: One of the control rules. It states that if the value of the attribute *edibility* is not known, then trigger the rule set *edibility-rules*.

The cross-compiler from KSS0 to BABYLON uses the weights of the attributes to sort the value-inference rules and the control rules, so that the inference process reflects the use of the weights in KSS0.

Currently, it is not possible to give a knowledge-level description of the processes (Karbach, Linster and Voß, 1990) within Hyper-KSE.

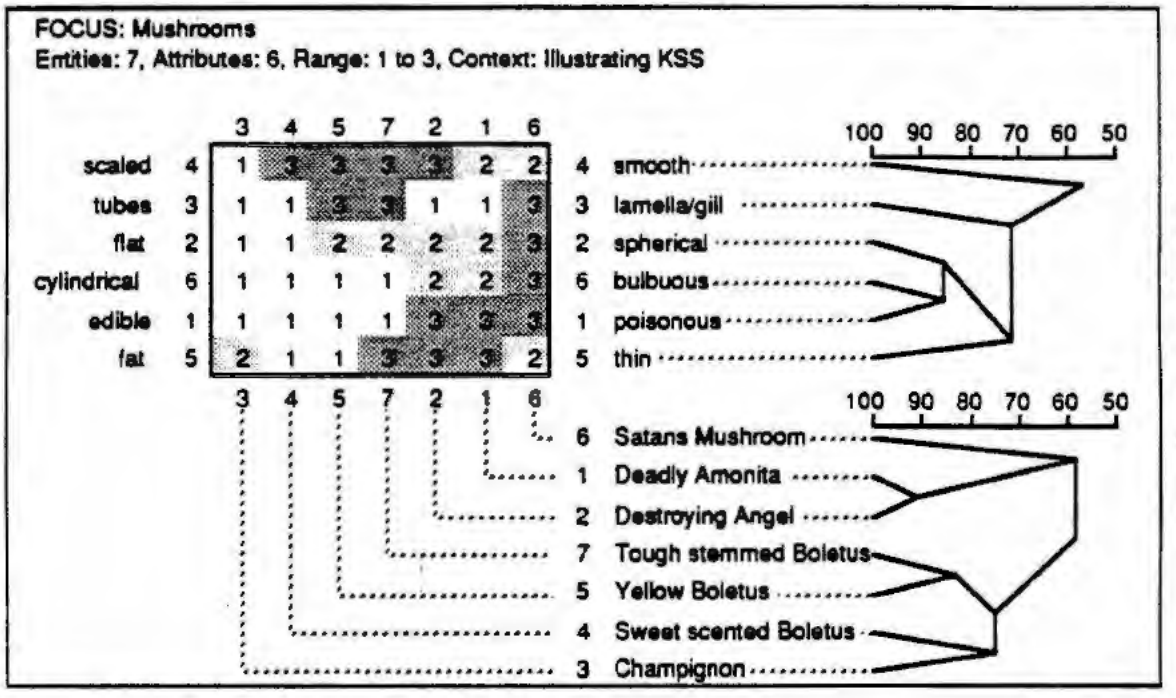


Fig. 6: Result of the cluster analysis using FOCUS in KSS0

5. THE USE OF THE INTEGRATION

We will illustrate the use and potentials of an integrated knowledge support environment with the development of a small knowledge-base on mushroom-classification. It is the goal of the system to infer edibility from a description of a mushroom. The ensuing knowledge based system is to be used by a non-expert, so it is necessary to explicate domain-specific terms to a non-specialist.

5.1. Eat it or Leave it ?

With KSS0 a knowledge base was developed that describes several mushrooms, e.g. *deadly amanita*, *champignon*, *yellow boletus* with discriminating attributes, e.g. *edibility*, *underside_of_cap*, *surface*, *stalk*, *rootshape* (see fig. 6).

The attribute *edibility* was defined as being an output attribute. With the help of INDUCT, an induction analysis algorithm of KSS0, rules of the following kind were generated:

```
...
Surface=velvety -> Edibility=poisonous 100% 18.4%
Rootshape=wrapped -> Edibility=poisonous 100% 18.4%
Underside of cap=tubes & Stalk=thin -> Edibility=poisonous 100% 18.4%
Stalk=thin -> Edibility=poisonous 66.67% 39.4%
Stalk=fat -> Edibility=edible 100% 32.7%
Underside of cap=lamella/gill & Surface=smooth -> Edibility=edible 100%
32.7%
...
```

During a consultation these rules can then be interpreted in BABYLON, to define the current value of the attribute *edibility* (see fig. 7).

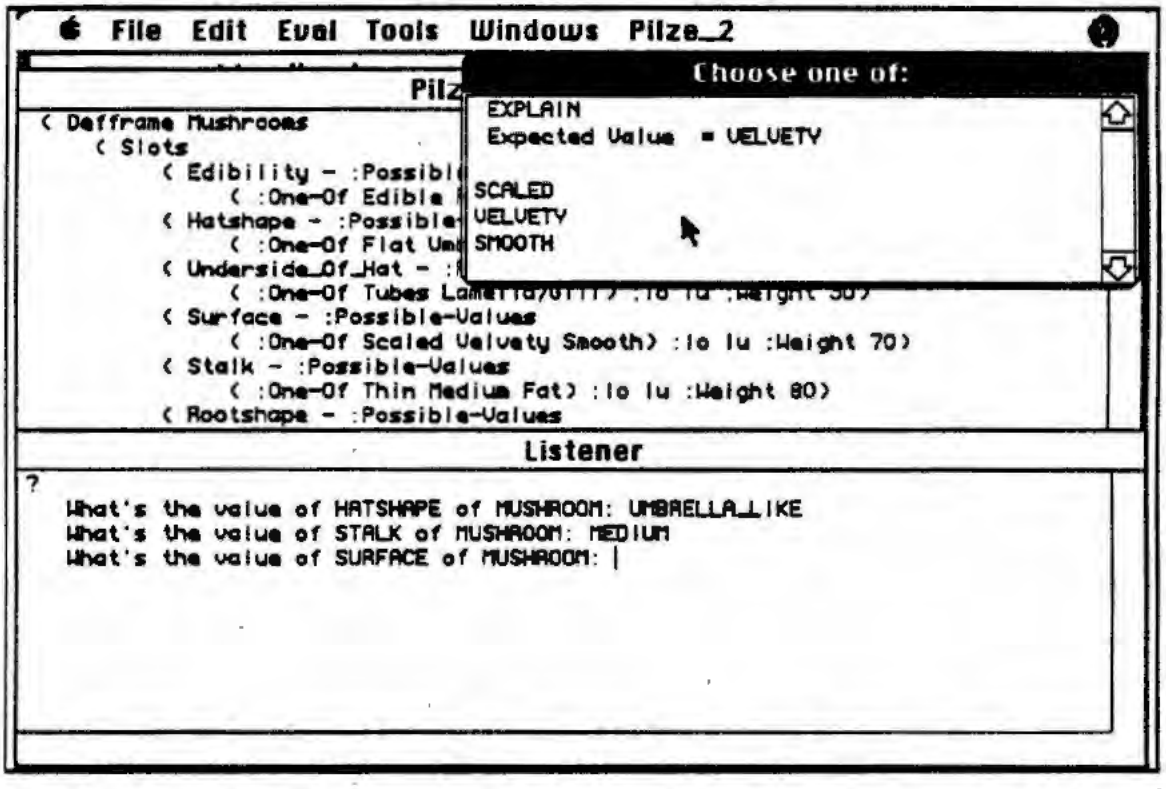


Fig. 7: BABYLON's query during a consultation of the knowledge base. The "EXPLAIN" choice accesses a description string in standard BABYLON or an annotation card if BABYLON is used within Hyper-KSE.

Mushroom classification systems have previously been developed and analyzed at GMD (Müller, 1986). One of the major problems with the use of these systems was the fact that to understand the system's queries for input, you had to be a domain expert, e.g. to answer *"Is the root bulbous, tuberous, cylindrical or wrapped into a skin?"* quite a bit of domain knowledge is required. Illustrations would allow the amateur to give adequate answers to these questions (see fig. 11). Other questions regarding color gradation, e.g. *"Is the color yellow, dark yellow, brownish or ochre?"* require even more sophisticated front ends to allow for reliable input.

5.2. Annotating and Accessing Knowledge Base Elements

As shown in fig. 7, a solely symbol-based interaction during a consultation requires expertise on the side of the client, to make sure that he/she has a conceptual model of the application task, such that he/she can understand the queries of the knowledge-based system. KSS0 provides facilities to annotate knowledge base elements during acquisition (see series of figs. 8, 9 & 10).

The KSS0-HyperCard link automatically creates an empty annotation card for every element of the KSS0 knowledge base. The card has room for text annotation(see fig. 9) and it has a link to another card (see fig. 10) that allows the input of other types of information, e.g. scanned pictures.

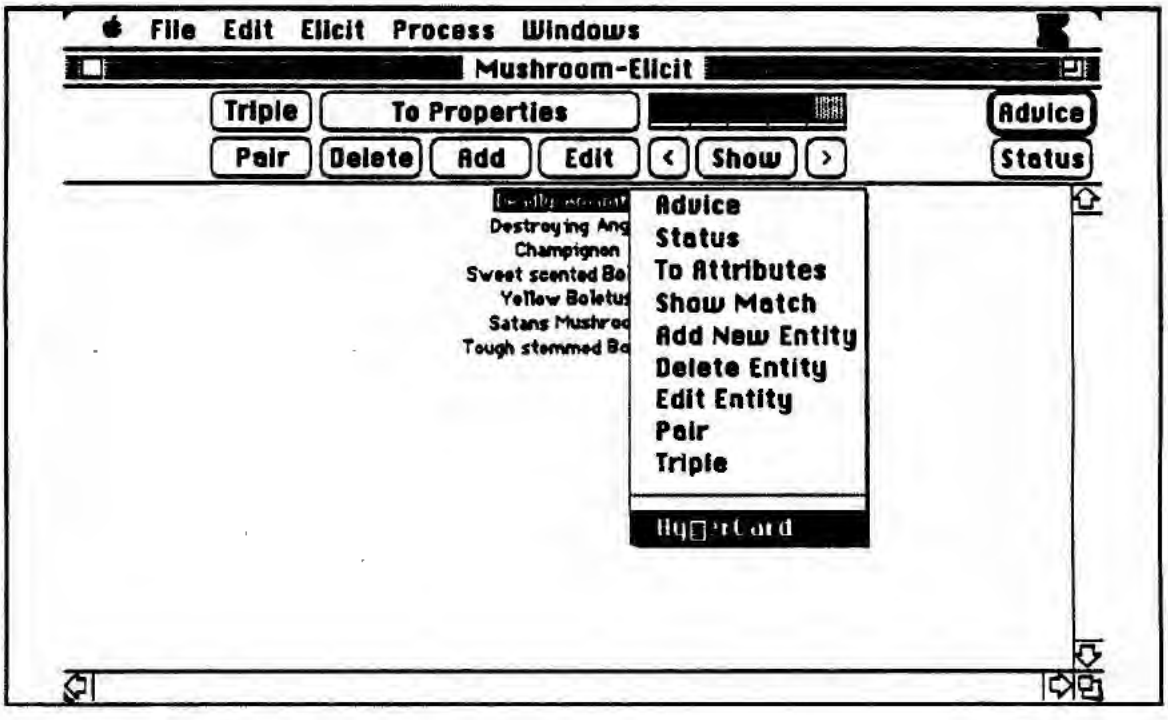


Fig. 8: Accessing HyperCard from KSS0.

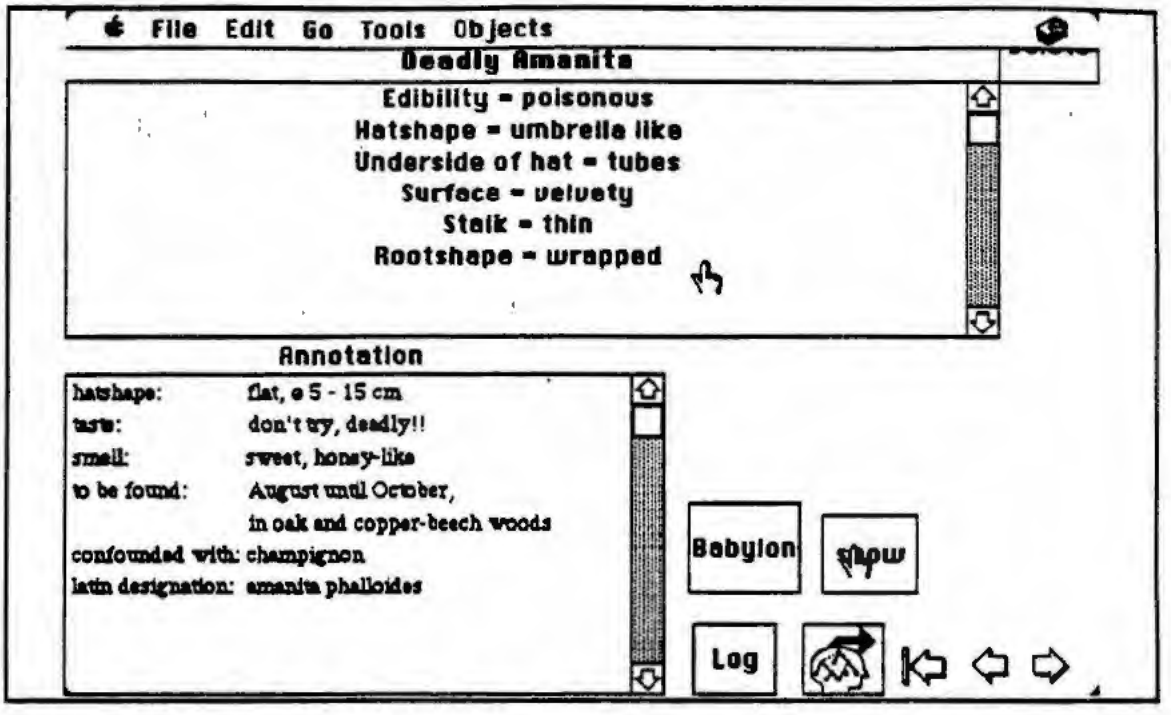


Fig. 9: A textual annotation of the concept mushroom "sweet scented boletus".

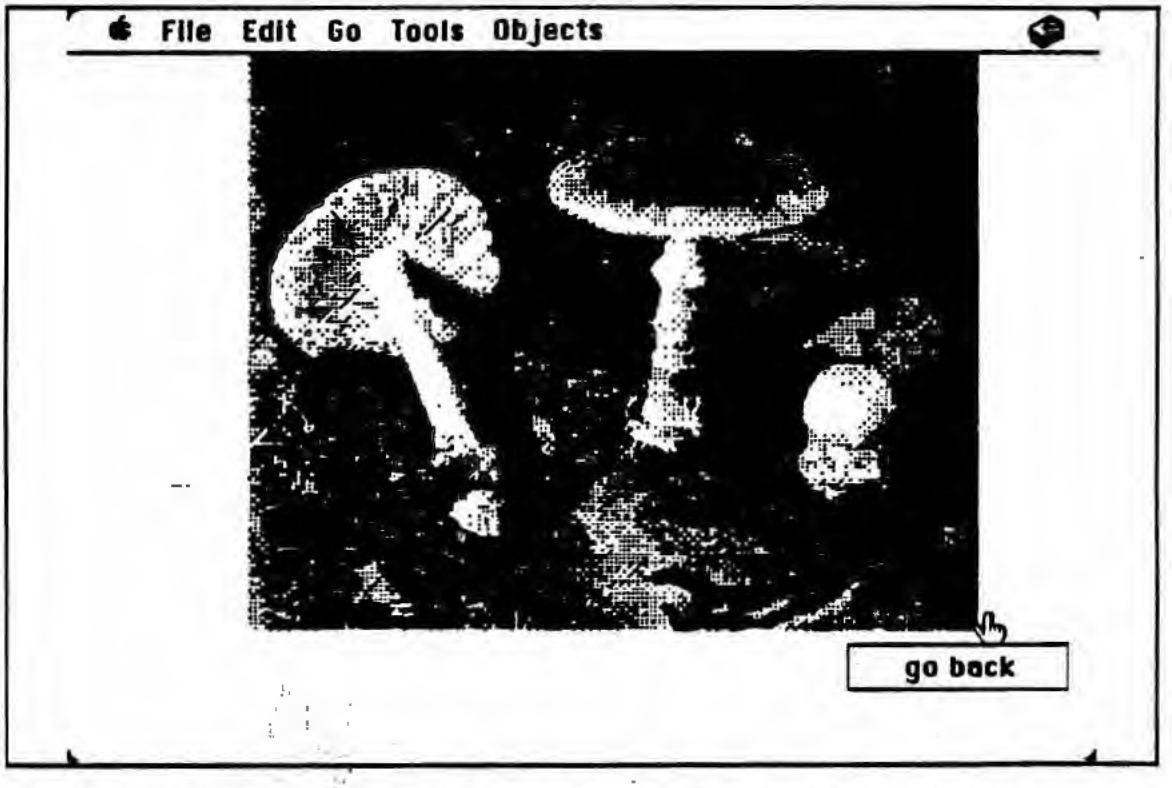


Fig. 10: In addition to the textual annotation other media, e.g. pictures can be used for explanation.

During a consultation of the knowledge base the annotation cards can be accessed through the BABYLON explanation facilities (see fig. 7) so that, if the client is confronted with unknown options, e.g. *"Is the ROOTSHAPE cylindrical, wrapped or bulbous ?"* he/she can then refer to the HyperCard annotations of these knowledge base elements to try to understand the query and the options (see fig 11).

During maintenance the standard knowledge base browsers of BABYLON allow direct access to the definitions of attributes and examples within the knowledge base, just as to the annotations initially defined in KSS0, so that the link between code and annotation can be followed easily.

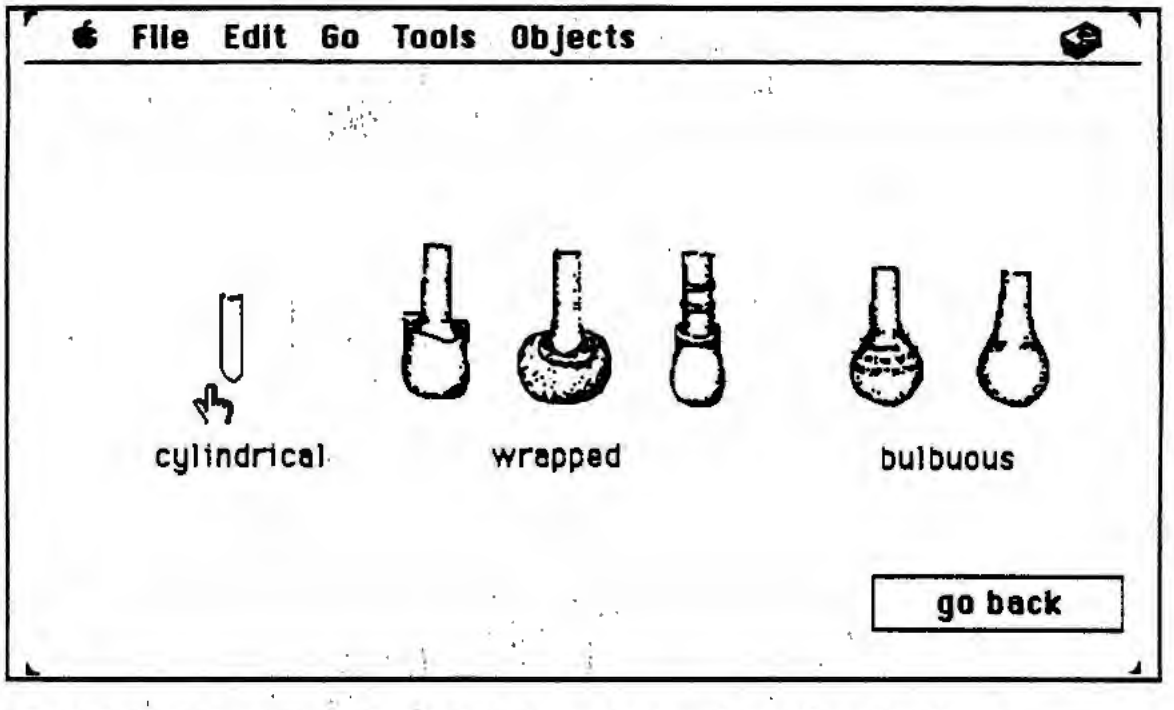


Fig. 11: Graphical illustrations in HyperCard of the possible values of the attribute *"Rootshape"*.

5.3. Incremental Development of the Knowledge Base

As mentioned above, exporting a KSS0 knowledge base into an operational environment does not only allow for consultation but also for validation and testing. During a consultation the user can access the HyperCard annotation of a knowledge-base element to get a better understanding for the meaning of the symbol. Using the analysis facilities of KSS0, such as FOCUS or PrinCom the client can view how the creator of the knowledge base used the knowledge base element in its context (see fig. 6). If needed he/she can then easily change or adjust the knowledge base. If the KSS0 knowledge base is modified, it will have to be transferred to BABYLON using the export mechanism before it can be used in a consultation (see fig. 3).

A special card allows for an easy access of the analysis and editing tools of KSS0 (see fig. 12) during a consultation.

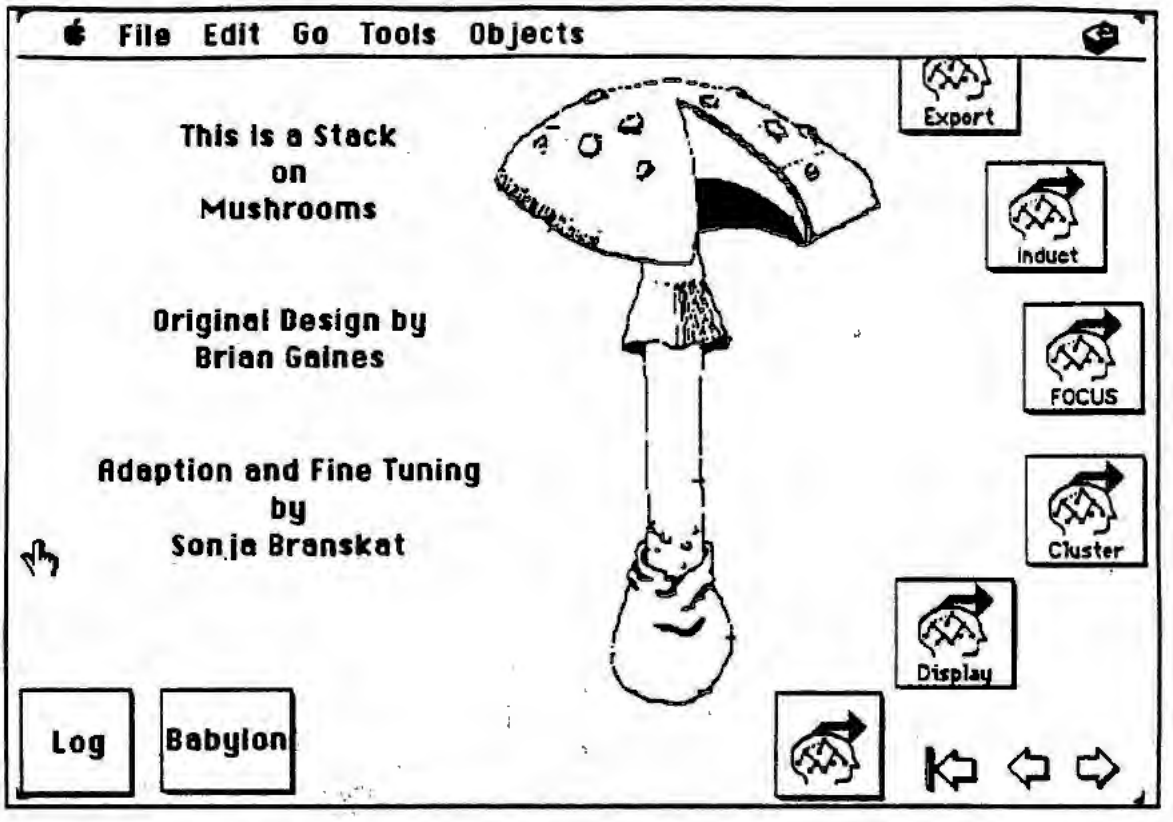


Fig. 12: A central card allows the easy access of the KSS0 analysis and editing facilities during a consultation in BABYLON

6. DISCUSSION OF THE HETEROGENEOUS INTEGRATION

The three tools that make up Hyper-KSE can all function independently, and have been used successfully on their own. They each have their own knowledge bases represented in their own formalisms, which within Hyper-KSE can be exchanged through an intercommunication protocol along *one-way streets*, i.e. a knowledge base may be edited in KSS0, but not in BABYLON, as this would put an end to the continued applicability of the acquisition tool in following extension/maintenance cycles. This means that in the current version of Hyper-KSE the acquisition tool defines the scope of the applicability of the integrated system.

The integration of different media in a heterogeneous environment allows for the transfer of knowledge between media, e.g. COGNOSYS knowledge bases (developed in HyperCard) could be formalized with KSS0 and interpreted in BABYLON. In such a case Hyper-KSE would allow for the browsing along the transfer-links between the media.

Formal consistency between knowledge base elements cannot be analyzed in Hyper-KSE, as a knowledge base is made up out of different media, i.e. formalisms (rules, clauses or frames) and virtually unconstrained documents (cards, picture sequences or sounds). Knowledge elements in different media are only linked through user-defined pointers and thus do not allow for consistency checks.

The heterogeneous integration requires a fraction of the effort required to build a comparable monolithic tool. The prototype of Hyper-KSE has been developed within a few person-weeks. As intercommunication protocols become more uniformly available, it will be more and more interesting to combine existing tools to reach qualitatively new functionality.

7. CONCLUSION

The experimental system described above shows the potential use of linking hypermedia and knowledge based systems, i.e. knowledge presentation in multi-media form is easier to understand and requires less domain-expertise from the customer than solely symbol-based presentation. Similar results have been shown in (Gaines and Vickers, 1987).

Besides allowing for multi-media annotation of single concepts, attributes and values the hypermedia facet of Hyper-KSE allows the integration of other existing or specifically created hyper-media documents, e.g. manuals, sketches or technical documents as integral parts of a knowledge based system.

At the same time the close integration of acquisition and interpretation allows for easy maintenance and analysis of the operational knowledge base. The client can access the annotations of the knowledge elements in HyperCard to understand novel terms, and he/she can use the KSSO analysis facilities to understand the use of a term within the knowledge base.

ACKNOWLEDGEMENTS

Thanks to Bernd. S. Müller who provided the material on mushrooms, to Sonja Branskat who enhanced the stacks to suit this application and to Hans-Werner Güsgen who gave very useful comments on earlier drafts.

REFERENCES

Christaller, DiPrimio and Voß, 1989. Christaller, T., di Primio, F. and Voß, A. *Die KI-Werkbank BABYLON*, Addison Wesley, Bonn (1989).

DiPrimio and Wittur, 1987. di Primio, F. and Wittur, K. BABYLON, a Meta-Interpretation Model for Handling Mixed Knowledge Representations. In *Proceedings of the 7th International Workshop on Expert Systems and Their Applications*, Avignon, 1987, pp. 821 - 833.

Gaines, 1989. Gaines, B.R. Knowledge Acquisition: The Continuum Linking Machine Learning and Expertise Transfer. In *Proceedings of EKAW89*, Boose, J.H., Gaines, B.R. and Ganascia, J.G., afcet, arc, eccai, Université Paris-Sud, Paris, 1989, pp. 90 - 101.

Gaines and Linster, 1990. Gaines, B. and Linster, M. Development of Second Generation Knowledge Acquisition Systems. In *Proceedings of EKAW90*, Boose, J.H., Gaines, B.R., Wielinga, B., Schreiber, G. and van Someren, M., University of Amsterdam, IOS Press, Amsterdam, 1990, pp. 143 - 160.

Gaines and Vickers, 1987. Gaines, B.R. and Vickers, J.N. Instantiation of Knowledge Structures in Books and Hypermedia. Tech. Rept. KSI/NML-87-BHM, Technical Report, University of Calgary, Knowledge Science Institute & Neuro-Motor Laboratory, Calgary, 1987.

Gappa, 1989. Gappa, U. CLASSIKA: A Knowledge Acquisition Tool for Use by Experts. In *Proceedings of KAW89*, Boose, J.H. and Gaines, B.R., Univrsity of Calgary, 1989, pp. 14/1-14/15.

Karbach and Linster, 1990. Karbach, W. and Linster, M. *Wissensakquisition für Expertensysteme: Techniken, Modelle und Softwarewerkzeuge*, Hanser Verlag, München (June 1990).

Karbach, Linster and Voß, 1990. Karbach, W., Linster, M. and Voß, A. Models of Problem-Solving: One Label - One Idea?. In *Proceedings of EKAW90*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G. and van Someren, M., University of Amsterdam, IOS Press , Amsterdam, February 1990, pp. 173 - 189.

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

Müller, 1986. Müller, B.S. Lehrmaterialien BABYLON: Die Beispielswissensbasen zur Pilzbestimmung. Tech. Rept. 221, Arbeitspapier der GMD, GMD, St. Augustin, September, 1986.

Musen, 1989. Musen, M.A. *Automated Generation of Model-Based Knowledge Acquisition Tools*, Pitman Publishing, London , Research Notes in Artificial Intelligence(1989).

Puppe, 1989. Puppe, F. *Einführung in die Expertensysteme*, Springer Verlag, Berlin (1989).

Shaw and Gaines, 1987. L.G.Shaw, M. and Gaines, B. An Interactive Knowledge Elicitation Technique Using Personal Construct Technology. In *Knowledge Acquisition for Expert Systems*. Plenum Press, Kidd, A., New York, 1987.

Tu et al, 1989. 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 (December 1989), 1439 - 1455.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Instructable systems

Bruce A. MacDonald
Knowledge Sciences Institute, Computer Science Department, University of Calgary
2500 University Drive, NW
Calgary, Alberta, Canada, T2N 1N4
bruce@cpsc.ucalgary.ca

ABSTRACT

Instructable systems — both instructable robots and instructable agents — must acquire skills and knowledge from examples and other instructions easily given by users in factories, laboratories and offices. Both senses of "instruct" are important: command and teach. The human interface must exploit the user's natural instruction abilities and require minimal acquisition of expertise prior to teaching. It is assumed that typical users will not be expert programmers, but will be able to do the tasks they wish to teach and also show them to other humans. Inductive learning techniques are employed to generalize the teacher's examples, in a manner biased by the teacher's other instructions, and thereby form a procedural task description. Instructions can drastically reduce the example and computational complexities of learning problems without compromising learnability. Existing machine learning systems are placed in an instructable framework. Three experimental prototypes are briefly described. Two systems instruct robots: one emphasizing examples and the other emphasizing more explicit instructions. The third is an instructable, office clerk metaphor. Instructability is seen as a small, but significant step toward intelligence.

1 INTRODUCTION

The eventual goal of my work is to produce intelligent systems that behave like "ideal" human workers in environments such as the factory, laboratory and office. They will be useful subordinates, taking orders and learning new tasks from human superiors.

At present it is just too difficult to design an intelligent system of this kind. We don't know enough about low-level abilities, such as manual dexterity, sensory processing, and legged mobility. Promising research has yet to mature in real world reasoning, learning to reason, acquisition of sequential tasks and procedural knowledge, and problem-solving in ill-defined environments. Most robot production is limited to mid and large sized batches where the effort and expense of setting up new tasks is warranted. It will be some time before a complete, integrated intelligent system is produced. In the meantime there is a disparity between the ambitious research in intelligent systems, and the comparatively limited abilities of factory, laboratory, and office systems.

This paper proposes an intermediate research goal of "instructable" systems which acquire tasks from a user's examples and additional instructions, but don't need other characteristics of intelligence. This is an appropriate methodology for investigating intelligence and I expect it to yield independently useful systems.

Two kinds of instructable system are possible: instructable robots which act in the real world, and metaphorical instructable agents which act in the human–computer interface. Both must acquire skills and knowledge from examples and instructions easily given by users. The system's human–computer interface must exploit the user's natural instruction abilities and require minimal acquisition of expertise prior to teaching. The paper begins with a broad view of "instructions," which includes computer programs as well as examples, cases, plans, rules, and indirect commands. Then a narrower approach is taken, by assuming users have little or no programming expertise, although they must be able to do the tasks they wish to teach and also teach them to other humans.

Section 2 motivates instructability and exposes it from several popular viewpoints, then discusses methods of instruction, user interactions, knowledge and skills required of a user, a qualitative measure for instructability, the special case of instruction by programming, and finally narrows the discussion to example–based instruction. Section 3 places instruction into exact and approximate learning models. Inductive learning techniques are employed to generalize the teacher's examples — in a manner biased by the teacher's additional instructions — and thereby form a procedural task description. Instructions can drastically reduce the example and computational complexities of learning problems, without compromising learnability. Example complexity is the number of examples required to learn a task. Computational complexity is the time and space complexity of the learning algorithms, and may depend on the number of examples and the size of the description space. Learnability measures the size of the class of learnable tasks. The notion of instructable systems extends these ideas to include the *difficulty* of providing examples (as well as the number), the instruction needed, and the prerequisite knowledge and skills. The example presentation order is also important, as is the system's ability to present its own examples for user approval. Instructable systems will encourage a user to present appropriately ordered examples, and will facilitate instruction by asking suitable questions.

Section 4 discusses metaphors for instructable agents, and suggests that these must be brought to "life" for the user, to the extent that users ascribe intentions to the agents. Three experimental Calgary prototypes are briefly described in section 5. The first is given instructions in the form of a simple, incomplete robot program, or concrete plan. This is completed during execution as the user guides examples of undefined movements, while the system automatically corrects for positioning errors. The second prototype learns robot assembly procedures from guided robot sequences. A spatial attention focusing mechanism constrains inductive search. The third is an automated office clerk which a user guides through task examples, giving instructions which aid learning. For instance, the user can explicitly direct the clerk's "focus" to a visible object on the screen. Section 6 places existing machine learning systems in an instructable framework, showing the broad relevance of instructable systems.

2 INSTRUCTABLE SYSTEMS

Instructable systems do not require all the characteristics of intelligence. There is no *a priori* need for such a system to have free–will, set its own top–level goals, be conscious, communicate in complex natural language, use common sense, or reason. Instead, we hold no particular expectations of intelligence, and only when necessary will we provide such characteristics.

The essence of instructable systems is both meanings of "instruct": command and teach. Primary requirements for them are to

- take simple verbal instructions to execute tasks
- acquire tasks from suitable interactions with humans, such as by being guided through examples and given simple kinds of help
- adapt to changing real world conditions (or the instruction would seem "brittle").

These abilities have been promised for some time in factories, laboratories and offices, in the claims made for programmable systems. However, the time to set up robot tasks, for example, makes this an inappropriate view; the systems are not easy to teach, program or instruct. Since they will do what they are told, instructable systems will be useful in their own right, even when the other abilities for intelligence are mastered by designers.

2.1 Motivating instructability

There are many areas where instructability exists in some form.

2.1.1 Human-computer interaction Instructability will be a major component of future human–computer research, since to be effective the capabilities of modern workstations must be more closely matched to those of humans. Future workstations will need to accept instructions in formats users wish to give.

2.1.2 Cognitive Science Norman (Norman 1980) has emphasized the importance of understanding all the parts and players of human interaction, in order to understand the whole. For example, the teacher must be modeled if the classroom situation of students is to be understood. This paper makes a similar claim, to understand machine learning one must also model the user and the interaction with the system.

2.1.3 Similarity–based learning It is interesting to consider instructability in the framework of a similarity–based learning algorithm such as candidate elimination (Mitchell 1982). What useful instructions can be given? If the examples and candidate concepts can be fully characterized by the designer, then no instructions are needed. But, more generally, a user might give information about the examples and possible concepts, as well as additional information such as the independence of features, which would be useful for factoring a version space (Genesereth & Nilsson 1987). This kind of instruction may be unnatural, unless users can express the biases in their own terms (e.g. by saying that new concepts about playing cards need not distinguish face cards). Instructions in similarity–based learning are examined in more detail in section 3.

2.1.4 Expert systems Requiring experts to articulate their knowledge is a bottleneck in knowledge–based system development (DeJong & Mooney 1986). The aim of instructable systems is to overcome this, by providing the right combination of human–computer interface and machine learning.

2.1.5 Explanation–based learning Instructions are like explanations in the sense of DeJong (DeJong & Mooney 1986). They don't provide proofs of examples, but they do provide useful explanatory information. Instructable systems must accept explanations expressed in forms appropriate for humans (i.e. instructions).

2.2 Methods of Instruction

There are various ways to transfer new tasks, as shown in table 1. Specifications, algorithms, programs, rules, and plans give explicit representations, while advice and comments are more indirect and demonstration methods are implicit. The choice of method depends on the nature of the system, the user's abilities, the task and the context. As you will see in the following sections, instructability relates to all four. For example, programmers normally choose to give instructions in the form of programs; while non–programmers don't. However, even programmers might want to teach a spray–painting robot using the common industrial technique of guiding it through an example.

Instructions may also be provided at different stages of a system's "life." Some will be built–in as primitives, such as "pick up," and common tasks, such as screwing in a threaded bolt. Other common tasks will be provided by the user before teaching a particular procedure. Some subtasks may not be taught until they are required *during* the teaching of some other task. Some subtasks may be named during teaching and taught afterward.

2.3 Interaction

Figure 1 shows a simplified instruction scenario. The user knows something about the task, and about instructing the task. The instructable system knows some other tasks, as well as something about acquisition. Constraints on teaching are assumed or enforced, such as producing simple examples before complex ones. The task is transferred by some form of interaction between the user and system. This model is very general, but will be developed in the following sections.

Method	Example
Specification	Digital filter response
Formal procedure	Digital filtering algorithm
Program	Sorting code
Abstract plan	Sub–goals in an assembly task
Concrete plan	"Grasp BOLT.1, Align BOLT.TIP with HOLE.3"
Rules	"Prescribe ABC when you see XYZ"
Advice	"Don't use crescent wrenches"
Comments	"Aluminum has no fatigue limit"
Examples	(Sequence of robot arm positions)
Show	(Watch a human assembling a motor)
Built–in	(Sequence to pick up an object)

Table 1: Methods for transferring new tasks

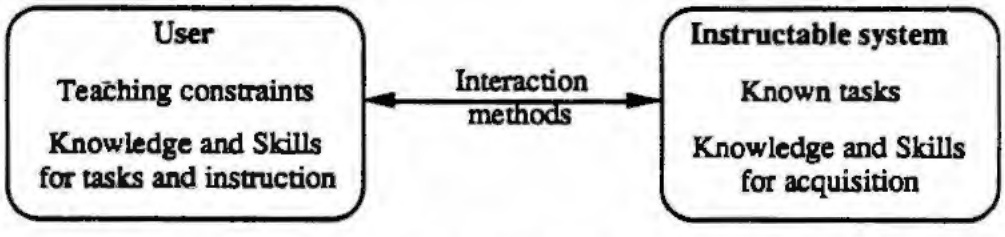


Figure 1: A user interacts with an instructable system so it can acquire the task

2.4 Knowledge and Skills

Let us consider the possible kinds of instructable system, before the next section narrows the discussion to non-programmers. Instructable systems must appropriately acquire knowledge and skills given the particular characteristics of the user, system, interface and task. The following aspects must be accounted for:

- The user's knowledge and skills for
  - task performance (the user must be able to do the task)
  - task understanding (the user must understand the workings of the task well enough to instruct it)
  - task instruction (the user must know how to instruct)
- The system's knowledge and skills
  - previously acquired tasks (may need to know subtasks and primitives)
  - inductive reasoning (must be able to generalize examples to concepts)
  - deductive reasoning (must be able to infer new facts from existing ones)
  - task modification and debugging (must allow the user to modify the task)
  - user interaction (must be able to communicate effectively with the user)
  - task representation (must be able to represent the structures required for the task)
- The interface characteristics of
  - physical load imposed on the user (e.g. back ache, eye strain, difficulty in lifting the robot arm)
  - cognitive load on the user (e.g. complexity of the task and teaching process)
  - method of transfer (e.g. leading, editing programs, visual programming)
  - assumed or enforced felicity conditions (e.g. examples must be provided in the right order)

There are two processes involved here. First, the user must acquire or already possess the abilities for performing the task and for understanding how the system must perform it. Second, the user must transfer the task to the system.

As an example, compare programming and guiding as methods for instructing a spray-painting robot. A factory spray-painter possesses the skills to perform the task, but not the programming ability. The appropriate method of instruction is for the robot to record the painter's movements as he or she paints with the robot's arm. The programmer must know something about spray-painting, and possess the skills and knowledge to explicitly specify and code the problem, but need not be competent enough to physically perform the task.

In both cases it seems reasonable to assume that acquiring the other's skills and knowledge would not be worthwhile. Also note that the spray-painter will find his or her job much easier than the programmer (who has to explicitly specify smooth, complex motions in space). However, the resulting task representation will likely be much less flexible in varying conditions than the program, and more difficult to modify.

Suppose the task to be taught is to sort some data efficiently. Sorting can be demonstrated, but to instruct an efficient sort the user will likely need a deeper understanding of the task, and the ability to communicate direct instructions to the system to achieve the required algorithm. Programming may be easier than giving examples. It is important not to suggest that demonstration and other styles of teaching replace all programming. Hybrid systems may enable most tasks to be demonstrated, with complex subtasks programmed.

Many popular research areas can be seen as attempting to improve instructability in particular ways, for example human-computer interface work aims to improve the transfer of information between user and machine. Table 2 shows different directions for instructable systems, when different assumptions are made about the user, system, interface and task. But how are we to select among these areas, and what does increased instructability mean?

Research area	*Instructable systems area*
Human–computer interaction	Process of task transfer
Programming language design	Matching users' and systems' prior knowledge and skills, assuming a proper training in computing science for the user
N'th generation languages	Minimize what the user has to know about the system, and build as much functionality into simple commands and constructs as possible (since the user knows something about simple algorithms, but little about complex system operations and complex abstract structures).
Software development tools and environments	Assuming a programmer is expert, improve the instructability by improving the programming method for task transfer
Make computer systems easier to learn about	Assuming a novice level of programming ability, improve the instructability. Allow more knowledge and skills to be acquired during interaction, so long as the amortized difficulty of this acquisition is minimized. That is, the effort of learning about the system should be more than offset by the improved performance in future interactions.
Design of robot programming languages for engineers	Assuming a factory engineer's knowledge and skills with sequencing machines and guiding robots, design a system that acquires robot tasks.
Guided robots	Assume a user has no programming ability, that fixed sequences are all that need be learned, and that the minimum of additional skills and knowledge is called for (being able to perform the task)

Table 2: Different directions for instructability.

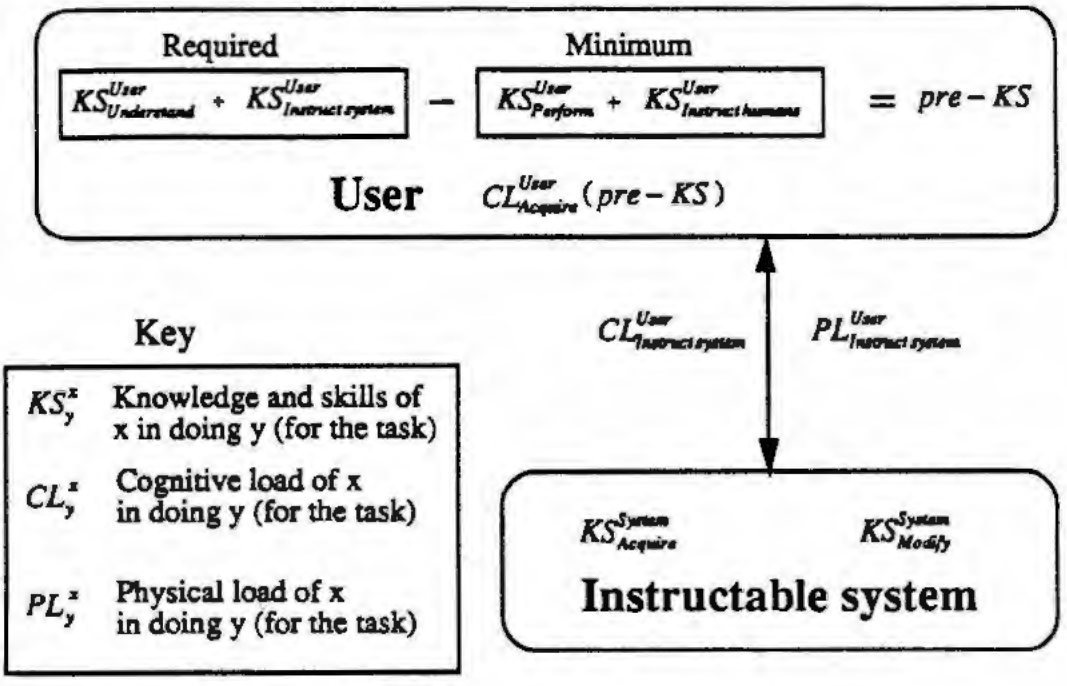


Figure 2: Cognitive and physical load, and knowledge and skill distribution during instruction. The user must acquire the additional knowledge and skills needed to communicate the task to the system — such as learning to program, or learning to gives rules to an expert system — incurring the cognitive penalty of $CL^{User}_{Acquire}(pre-KS)$. Then the task must be taught, with the physical (e.g. weight of the robot's arm) and cognitive (e.g. solving decision problems) effort in instruction ($CL^{User}_{Instruct\ System}$ and $PL^{User}_{Instruct\ System}$).

2.5 Instructability

What makes a system more instructable? Figure 2 displays the distribution of load, knowledge and skill between a user and system. The difference between the user's existing knowledge and skills, and those required for instruction, must be minimized, as the user must make up the difference. This can be unreasonable; an office worker would not want to learn Pascal just to instruct a new tax calculation to a machine. There are also cognitive and physical difficulties in instruction itself. Figure 3 postulates a rough measure of instructability which includes the ratio of task complexity to difficulty of instruction (prerequisite training plus the process of instruction). This is discussed in the next section.

2.6 Programming vs guiding

One research direction is to improve programming languages to be more suitable for explicit specification by humans. But my aim is to increase the power of methods such as guiding, to overcome their inflexibility, without requiring extensive prior knowledge and skills, except those most humans have anyway. So the idea is to minimize the additional knowledge and skills ($pre - KS$ in Figure 3) needed in order to acquire any practical task. One exception might seem to be where there are educational goals as well, since that goal is to *maximize* the new knowledge and skills acquired during instruction of a system. But in general this can be seen as minimizing the cognitive load of instruction, on average, over a long period (ie amortized cost).

Learning cannot be achieved by examples alone, as this is too difficult (See section 3). The danger is that the complexity of programming will be transferred to teaching, which is a less direct method of instruction, and therefore more prone to causing users as much if not more difficulty than programming does. But programming *must* be avoided when users are non-programmers. The danger is avoided by using other

$$Instructability = \frac{Task\ Complexity}{CL^{User}_{Acquire}(pre - KS) + CL^{User}_{Instruct\ system} + PL^{User}_{Instruct\ system}} + Learnability + Computational\ Complexity$$

Figure 3: Measure of instructability. Instructability includes the difficulty of preparing for and instructing a task, plus the measures for computational complexity (speed and space for the computation) and learnability (number of learnable concepts). The number of examples is included in the topmost measure

instruction methods as well as examples. So the complexity of programming is transferred to areas of knowledge and skill *already* possessed by humans.

The cost of acquiring generally useful skills and knowledge such as programming will be amortized over all programmed tasks. But this does not alleviate all difficulties of programming. The main difficulty is the need to explicitly specify tasks. The problem lies in translating knowledge and skills into explicit algorithms and programs. Humans are just not very good at this, even when they have been professionally trained.

A programmer must also deal with tricky details of syntax and semantics (and operating systems). Some of these difficulties will be overcome by improved programming languages. However, some are inherent to programming real world tasks. Declarations about the world can explode. Simultaneous many–dimensional programming may be cognitively impossible for humans. Sensory information processing is limited and incorporating this into procedures difficult, at least partly because of the lack of computing abstractions suitable for expressing interactions with the real world.[1] Sensing requires parallel asynchronous processing of unpredictable information, and coordinating this with prespecified action sequences.

2.7 Instruction by examples

Instructable systems currently rely heavily on the existing work in example–based learning. As methods for instruction are improved examples will not be so predominant. The next section examines learning from examples both with and without instructions.

3 INSTRUCTION REDUCES COMPLEXITY

This section examines how instructions reduce the number of examples and the computational complexity, without compromising learnability. Generally an example–based instructable system can be seen as accepting several traces of behavior which all belong to some — yet unknown — procedure. We can think of this procedure in graphical form, such as a finite, recursive or augmented transition network, depending on the

[1] The fashion of stateless programming (e.g. in functional programming languages) cannot be followed as the state of the real world must be explicitly represented.

sophistication of the internal language for procedures.[2] The problem then is to produce a "good," compatible procedure graph of the required type, given some example graph paths.[3] In this section we abstract away from the sequential nature of this problem, and examine how static concepts, such as those represented by boolean n-tuples, can be learned from examples *and* other instructions.

An instructable system reduces the amount of data needed by an induction algorithm in order for it to learn a description. There are two components: the information communicated and any additional information *about* the communication process (such as common abbreviations).[4] The system may exploit the user's natural interaction and instruction skill so that little or no extra information is required prior to instruction. There is a net reduction in transferred information since the system knows about and taps the user's existing abilities, or models.

Instructions reduce the number of examples required, by biasing the candidate concept space. This section will present examples of this benefit from instructions, in exact and approximate learning paradigms. More formally one is concerned with complexity in learning (number of examples and computational complexity of induction) and computability in learning (learnability). For the moment we will leave aside other measures of instructability.

Instruction is effective when the complexity is reduced without compromising learnability. Instruction does actually reduce the class of learnable concepts, but in effective instruction, only concepts that are not wanted will be eliminated, so learnability is not reduced.

3.1 Boolean n-tuples

Suppose we wish to instruct an office system in a sequential task that manipulates windows. A conditional branch may be required, and the branch condition may be based on something visible in a window. I will assume windows are described by n Boolean features, such as **front?**, **resizable?**, **empty?**, **hastext?**, and so on. Then there are 2^n possible examples, and 2^{2^n} possible subsets, each of which corresponds to a different window description. In general to learn a description exactly one needs to have seen *all* 2^n possible examples classified for membership in the concept, since the aim is to select a single subset amongst the rest.

Biasing the description language can reduce the description space, required number of examples, and the complexity. For instance, suppose the description sought is an exact conjunction. A standard technique is to represent the descriptions in a partial order — a version space — on "more general than" (set inclusion of extensions) and collapse upper and lower boundaries of this space using the candidate elimination algorithm (Mitchell 1982). The number of examples needed here depends on which examples are given, but may still be a considerable number.

The minimum number of examples to learn a conjunct of m features is two positive ones, and every one of the possible m "near misses" as negative examples. A near miss has just one feature wrong. So you need something like the log of the number of possible descriptions as the number of examples, since m cannot exceed n. But this is much fewer than 2^n examples of the general case. The number of possible descriptions is 3^n; each element may be 0, 1 or irrelevant (written "?") and on the order of only n examples are needed. See Figure 4. So one simple form of instruction is for the user to specify that descriptions be conjunctions of features. This might seem unlikely to be a "natural" instruction from a human user, but one can readily conceive of an interface that asks a user to identify visible window parameters, then asks whether there are any other quite different examples (No ≡ "conjuncts will do").

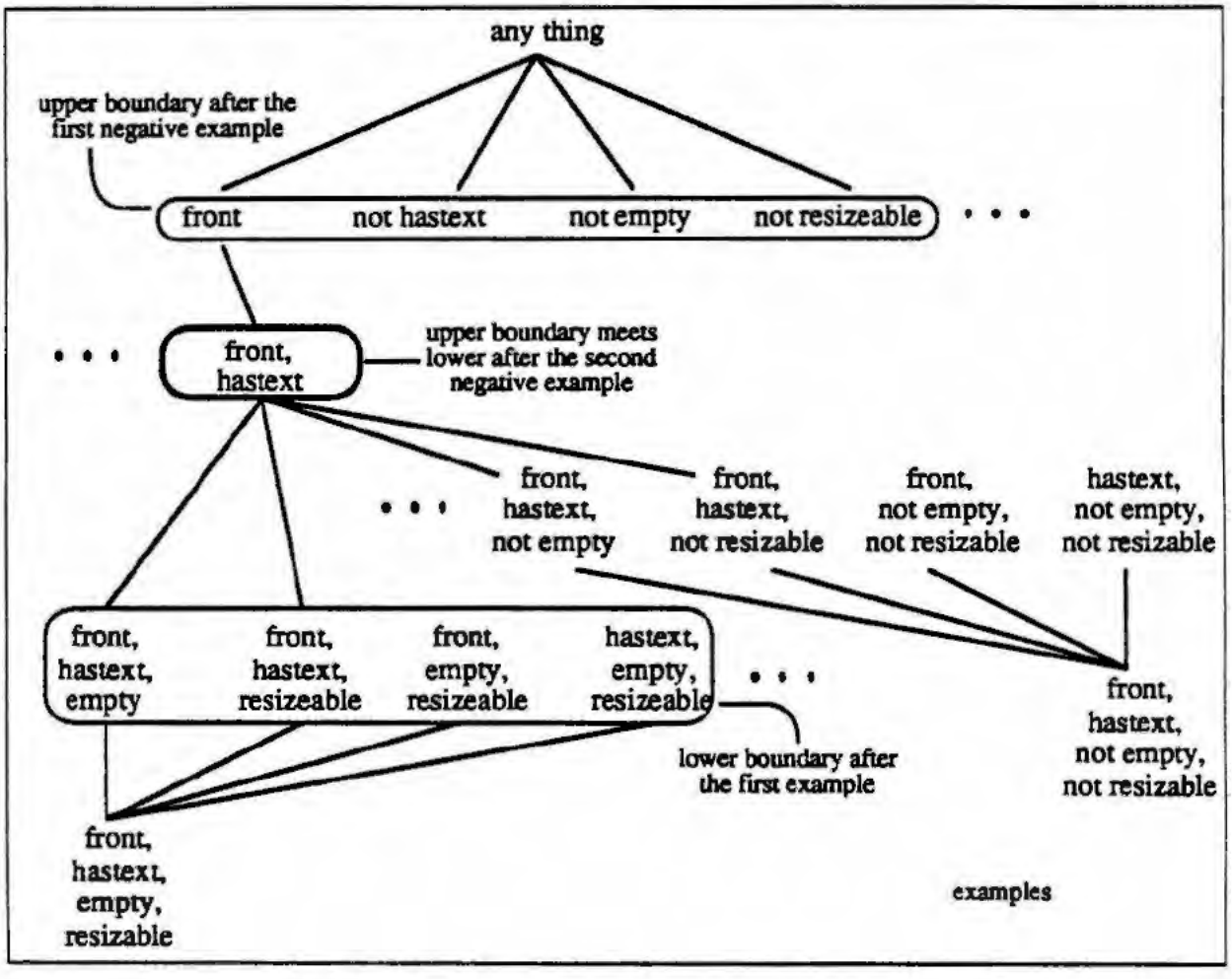
The problem of learning a multiplexor function from examples illustrates the power of instructions.[5] The

[2] Briefly, finite networks allow conditionals and non-counted loops, recursive networks allow parameterless subprocedures, and augmented networks allow variables, subprocedures with parameters, and are Turing equivalent (Barr & Feigenbaum 1981). All these programming constructs are important, but there isn't space to deal with them here.

[3] Normally one assumes that the collections of traces traverse every graph arc we wish to induce. So the problem is a finite one, limited by the number of nodes in the traces. But still the complexity is horrendous, since later nodes may depend on variables initialized in earlier nodes.

[4] Similar to the trade off between the model size and the efficiency of communicating encoded data.

[5] Robin Cockett's "knowledge shaping" of the multiplexor problem for decision trees (Cockett & Herrera 1990) suggested this



Example concept to be learned: **front, hastext**

- positive examples
  front, hastext, empty, resizable
  front, hastext, not empty, not resizable
- negative examples
  not front, hastext, empty, resizable
  front, not hastext, empty, resizable

Figure 4: A sample version space and learning problem

system must learn to use the address bits of the multiplexor to select the appropriate input feature for output. The worthwhile instruction is "focus on the address bits," so that all descriptions include address bit values. Suppose there are two binary address bits and therefore four binary data inputs, to give six features. Then there are 64 examples and 2^{64} possible descriptions. However, if we insist every description specifies address bit values, the description space is greatly reduced.

3.2 Approximate learning

Approximate learning, which occurs in the full 2^{2^n} space, allows error ϵ — the maximum fraction of examples misclassified by the description, in $(1-\delta)$ of the training sets — and the number of examples required m is roughly $1/n$ times the log of the number of descriptions (Dietterich 1989).

Also m must be at least

$$\frac{VCdim(H)-1}{32\epsilon}$$

where H is the hypothesis (description) set and $VCdim(H)$ is the Vapnik–Chervonenkis dimension of H, which represents the way the space structure is divided. Assuming $VCdim(H)$ at least 2, ϵ between 0 and 1/8, and δ between 0 and 1/100, then $VCdim(H)$ is at least

$$\frac{1}{n+2}\log|H|$$

So then the number of examples needed in a boolean feature space is on the order of $2^n/n$. Thus by accepting an approximate solution, concepts can be learned with fewer examples, although this effectively reduces the number of concepts that can be learned (concepts that differ only a little will not all be represented in the learnable set). Unfortunately, there does not seem to be a way to use instructions to select the descriptions that are ruled out of consideration in approximate learning (but perhaps this might be achieved by having the user specify how many features may be incorrect in the learned concept).

3.3 Instructions

Now if the user is able to focus the system's attention on the required features, and we assume a conjunction is required, then only one example is needed, giving the feature conjunct. The user is supplying a kind of non–verbal explanation. The system knows enough to learn from a single example, so we can see the result as a kind of explanation–based learning in the sense of DeJong (DeJong & Mooney 1986). The office clerk in section 5 provides an interactive dialog in which a user may instruct the clerk to focus on parts of the screen. If something more complex than a conjunction of features is required, then more examples will be needed, with the user focusing on a conjunction in each. The system must combine these to form the required description, such as

(front ∧ resizable) ∨ hastext,

using a method such as A^q (Falkenhain & Michalski 1986, Michalski & Chilausky 1980).

So instruction can drastically reduce the number of examples and the computation required when inducing a description, without sacrificing the learnability. It is not clear though whether the expectations of "focusing" may be too strong, although the clerk metaphor described below shows the feasibility of focusing on simple features.

---

example (personal communication).

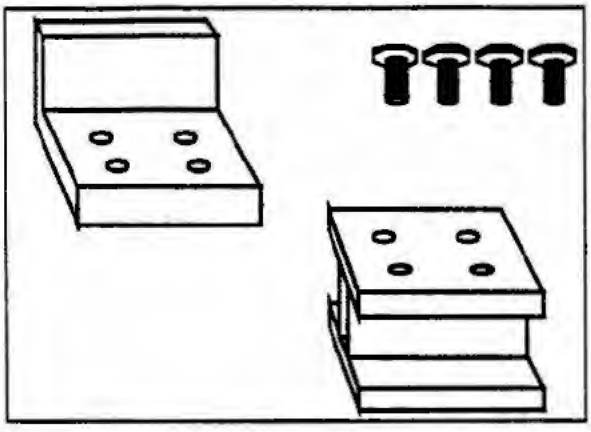


Figure 5: An example "bracket" task for ART

4 LIVING METAPHORS

A significant question remains about measures of instructability. How do we know what kind of instruction is best? How do we improve instructability? What is the minimum acceptable level of instructability?

To what extent must the designers tap users' human interaction abilities? There is a point which must be reached, beyond which the system becomes animated, not in the graphical sense, but in metaphorically coming to "life." Once this point is reached users will feel comfortable about new interactions with the system. This is a difficult notion to analyze for a designer. However, much of it would be achieved if users took Dennett's (Dennett 1981, Dennett 1987) intentional stance with the system. Dennett also presents the physical and design stances, which correspond to interaction at the physical level, and functional level. The intentional level requires the user to ascribe intentions and purposeful behavior to the system, something Dennett sees as essential if humans are to effectively interact. This is achieved partly by providing a suitable human metaphor in the office system, where there isn't a real robot to take an intentional stance with.

5 CASE STUDIES

5.1 ART (Pauli 1990)

Let us assume a robot factory engineer has experience with numerically controlled machines, and understands the notion of an algorithm. ART requires him or her to give an incomplete program, specifying only what is straightforward. At execution time, the user helps the system complete the program, mainly by guiding examples of individual actions. Errors resulting from the mismatch between program and guiding are automatically corrected.

Figure 5 shows a task for ART, to put the bracket on the beam and screw in four bolts. Figure 6 shows the incomplete program first specified by the user. It appears complete. However, the object positions are not defined, the order of bolt insertion is undefined, and the orientation of the bracket on the beam is left open. ART defers the user's commitment to these until an example can be given. The program is really a concrete plan for the task, with the details left open. At run time, the system asks the user to guide any unknown movements, then uses the resulting arm position to calculate the object position so the movement can be automated.

ART's language as implemented is somewhat limited, for example it lacks procedure calls. However, it is rich in expressing manipulations, and since it is based on another robot programming language, PAL

```
task affix-bracket

model "pauli/art/ext/progs/bracket
int   i
trans j

begin
  moveto BRACKET.GRASP
  grasp
  align BRACKET.HOLE.i with BEAM.HOLE.j
  release
  for i = 1 to 4
   begin
     moveto BOLT.GRASP
     grasp
     align BOLT.TIP with BEAM.HOLE.j
     screw
     release
   end
end
```

Figure 6: The ART program for the bracket task (from (Pauli 1990)).

(Takase *et al* 1981), extensions to typical language constructs should not be difficult. However, while the addition of conditionals will not be difficult, their proper incorporation with the robot will require a careful treatment of the asynchronous nature of sensory information in the sequential program.

ART's language improves on the PAL system by significantly reducing the code a user must produce, and making this a description much nearer a natural language description of the assembly task. It does this by automating much of the underlying robot control, using language constructs for movement and alignment, and by deferring some of the user's decisions until execution time when they can be more easily made.

During execution the user gives examples to complete the program, and the system corrects most user errors as the task is completed. The user may have the arm release an object, manually reposition the object or even perform an assembly step such as screwing in a bolt, then guide the arm to regrasp the object in the new position. Counted loops are unordered, allowing the order to be determined at run time. ART operation is summarized in Figure 7.

It seems a system such as ART must defer commitment in order to form an appropriate interface with the user. The user doesn't have to make decisions until they really need to be committed, and only then if they can't be automated.

In the future, machine learning techniques might be added to ART to learn about fastening and mating conditions. The assembly task knowledge implicit in ART's operation should be made explicit.

5.2 Office Clerk (Campbell 1990)

The direct manipulation, desktop metaphor does not provide an appropriate method of programming (the user must "drop back" to the level of the development system when writing a program). A *programming metaphor* is needed. This system provides an "office clerk" that is a metaphor of a real office assistant, and is depicted in Figure 8. This prototype version of the office clerk allows the user to manipulate electronic mail boxes and messages, and instruct the clerk to do tasks with these, by giving examples and instructions.

While the user guides the clerk through a task, mouse and menu operations are recorded. The user may

1. Assume there are fastening operations (needed for error correction)
2. A description of each object is provided by the user from computer generated design data
3. User writes an (incomplete) program in ART
4. ART compiles the program, using the object descriptions, and including code to interact with the teacher at run time to complete the program
5. ART attempts to execute the program
6. ART asks for user guidance when a transform is undefined (but only for operations where there are enough visual cues to facilitate the user's guiding)
7. The user positions the arm
8. ART reads the position, calculates the unknown transforms if it can and remembers the rest until more information is available
9. Errors corrected at enforced accuracy points

Figure 7: ART's operation

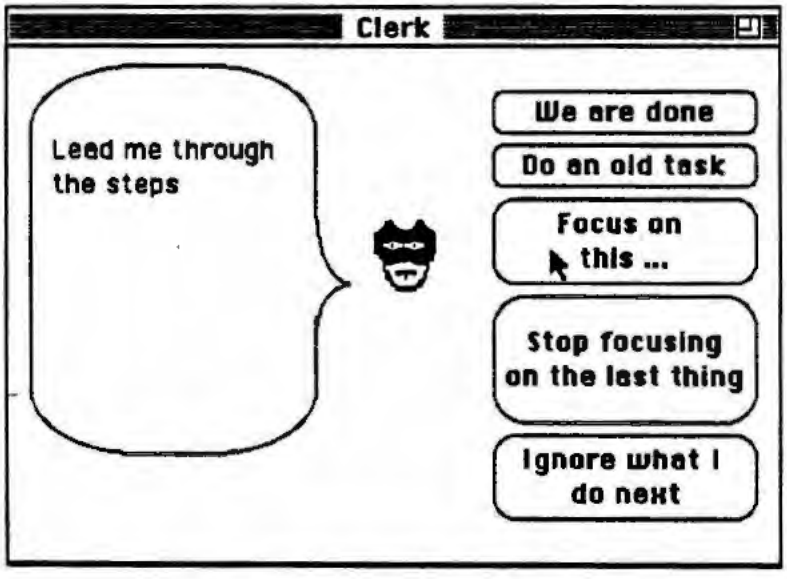


Figure 8: The office clerk metaphor (Campbell 1990). The clerk responds in the left hand "speech bubble," and the user may select a bubble from the right.

instruct the clerk to focus on a particular visible attribute, and this causes a conditional branch to be formed. Conditional branches can be nested. The task can be interrupted and debugged by stepping through the recorded actions, and this also happens when a conditional doesn't match, allowing the user to guide an additional branch. Window parameters are recorded during instruction and may be generalized (by candidate elimination on the window concept) as more examples are given of the same task, so that window objects in a task become variables. These features are recorded: window type (mailbox or message), parent window, index in mailbox list, and name. The learning is interactive so that a teacher may confirm inductive generalizations.[6]

The clerk gives the user an opportunity to give instructions to something tangible, rather than the mysterious internals of machines which, for example, automatic script writers must address.

5.3 ETAR (Heise 1989, Heise & MacDonald 1989a, Heise & MacDonald 1989b)

ETAR learns robot assembly tasks — such as stacking blocks — from user-guided examples. No other instructions are given. An automatic focusing mechanism limits the computation required. ETAR takes raw sampled robot position data, and a world model[7] and:

- Annotates the robot data with focus of attention information, about the objects nearby the hand
- Partitions the numerical robot data into symbolic robot actions (such as move-to), using the focus of attention to aid partitioning
- Generalizes the symbolic sequence to find loops
- Matches sequences to find conditional branches
- Generates the arguments for symbolic robot actions, which may be functions of characteristics of objects being manipulated
- Finds the conditions for loop exits and branches

I expect ETAR will eventually need instructions if it is to learn more complex tasks. Even in simple tasks the induction of functional relationships in the sequence is prohibitively expensive. The project emphasizes examples, and the goal is to give ETAR as much automatic "focusing" as possible, before adding instruction to the design.

6 RELATED WORK

(Halbert 1984) presents an office system in which a user guides examples and gives instructions to explain references to objects in the office metaphor. If the user selects a file icon during guiding the system asks the user to indicate what this means. Possibilities might be the object with that name, every object in that folder, or whether the system should always ask the user to specify the object at run time.

(Nix 1983, Nix 1984) describe methods for programming text manipulations by example. He argues that a user's execution trace in text editing will not be particularly helpful in inducing a program, since there will be much variability and inconsistency in the traces. So, he records only the input and output, and induces a "gap" grammar to perform the translation.

(Myers 1988) examines problems with programming by example, finding it unsuccessful for reasons including:

- Choosing examples is too difficult for users. Since examples give the system no guidance about the structure of a program, exactly the right examples must be chosen.
- It is hard for users to specify what data a procedure operates on (examples show operations, but don't really describe data).

[6] It might be less intrusive for the system to automatically generalise and leave the user to interrupt when a mistake is made.
[7] This could be provided by sensory information.

- Editing is difficult.
- Control structures are hard to communicate.

Myers avoids some of these problems by building in considerable knowledge about the learning domain — graphical interfaces — and designing a powerful language for users to give commands in. The user is expected to know something about programming to do this. The system can be taught interfaces much more quickly than they can be programmed. I see this as an instructable system that assumes users have some simple notions of graphical interfaces, and straightforward programs.

Instructable systems can avoid these pitfalls of programming by example, since instructions can directly indicate structure, specify data for operations, help with editing, and specify control structures. In addition, by the appropriate selection of metaphors for programming — see the office clerk above — a designer can make the instruction more acceptable to users.

(Andreae 1984a, Andreae 1984b) describe a 2–D robot simulation that learns robot procedures from examples. It induces loops, conditionals, and functional dependencies somewhat similarly to (Heise 1989), but without focusing, and with symbolic input from the user.

(Segre 1988) describes a robot control system that has considerable domain knowledge for assembly. The user guides it through an example to show how a specified goal may be reached. The system combines the example with its knowledge to form a description of the task.

(MacDonald & Witten 1987) describe a method of giving verbal corrections to a robot on the fly. This kind of dynamic interaction will be increasingly important for instructable robots.

(Maulsby *et al* 1989, Maulsby 1988) present a system for programming graphical constructions by example. It is assumed the user has some knowledge of graphical constructions. The user guides examples, and the system attempts to induce conditionals and loops, and so predict future actions.

(Bareiss 1989) describes the PROTOS system which learns concepts by remembering important examples (exemplars) and important relations amongst them. PROTOS interacts with the user to determine the significant characteristics, in a reasonably natural way. It is an interesting example of an instructable system since the representation (based on exemplars as opposed to a concept description language), and interaction techniques seem well–suited to human abilities.

(Hammond 1989) describes a case–based planner. There is a heavy emphasis on learning since planning is seen as reusing and learning from experience, so that new plans can be built and past errors avoided. Past plans are remembered, including those that fail by not reaching the required goals (they might be useful for avoiding those goals in future), and those that achieve the wrong goals (they should be avoided in the future if those goals are unwanted).

Both Bariess's and Hammond's systems employ considerable indexing into memory from the input information. This seems particularly suitable for learning systems, where the formation of associations leads naturally to indexed plans, goals, and exemplars. The recent interest in situation actions and reactive planning (Clancey in press, Agre 1988) is similarly interesting and I expect there will more work that brings together reactive reasoning, exemplar based acquisition, and learning from examples and instructions.

Knowledge acquisition in practical situations for expert systems must consider the importance of instructability. Some work already addresses this. In addition to the systems mentioned above, knowledge elicitation techniques, such as (Shaw 1980, Shaw & Gaines 1989), build interfaces *based on* the users' knowledge, a kind of reflexive instructable system. For example, the system might extract the important distinctions a user makes, and as a result change its interaction with the user to adopt those distinctions.

7 CONCLUSION

Instructable systems for non–programmers are a worthwhile research goal, combining machine learning and human–computer interaction to form systems that can acquire knowledge and skills from humans. Inductive techniques alleviate difficulties in the human–computer interface, while a suitable interface reduces the requirements of induction. Appropriate instruction methods can overcome the complexity of induction without sacrificing learnability.

ACKNOWLEDGEMENTS

The Knowledge Sciences Laboratory at Calgary, and particularly Brian Gaines and Peter Andreae (while visiting) contributed to discussions on the nature of instructable systems. Rosanna Heise, David Pauli and Bruce Campbell worked extremely hard to produce prototype instructable systems. Jacky Baltes, Rosanna Heise, John Lewis and Ian Witten commented on a draft. The Canadian Natural Science and Engineering Research Council funded the research.

REFERENCES

Agre, P. E. (1988) *The dynamic structure of everyday life.* Phd, MIT Dept of Electrical Engineering and Computer Science, October.

Andreae, P. M. (1984a) Constraint limited generalization: acquiring procedures from examples. In *Proc. American Association on Artificial Intelligence*, Austin, TX, August.

Andreae, P. M. (1984b) *Justified generalization: acquiring procedures from examples.* PhD thesis, MIT, Dept, of Engineering and Computer Science.

Bareiss, R. (1989) *Exemplar–based knowledge acquisition.* Academic Press.

Barr, A. and Feigenbaum, E. A. (1981), editors. *The handbook of artificial intelligence*, volume I. Addison Wesley.

Campbell, B. (1990) Office clerk II. Final year project report, University of Calgary, April.

Clancey, W. J. (in press) The frame of reference problem in the design of intelligent machines. In vanLehn, K. and Newell, A., editors, *Architectures for Intelligence: The 22nd Carnegie Symp. on Cognition.* Lawrence Erlbaum Associates.

Cockett, J. R. B. and Herrera, J. A. (1990) Decision tree reduction. Manuscript, April.

DeJong, G. and Mooney, R. (1986) Explanation-based learning: an alternative view. *Machine Learning*, 1(2):145–176.

Dennett, D. C. (1981) *Brainstorms.* Harvester Press, Brighton, Sussex.

Dennett, D. C. (1987) *The intentional stance.* MIT Press.

Dietterich, T. G. (1989) Limitations on inductive learning. In *Proceedings of the 6th Workshop on Machine Learning.*

Falkenhainer, B. C. and Michalski, R. S. (1986) Integrating quantitative and qualitative discovery: The ABACUS system. *Machine Learning*, 1:367–401.

Genesereth, M. R. and Nilsson, N. J. (1987) *Logical foundations of artificial intelligence.* Morgan Kauffman.

Halbert, D. C. (1984) Programming by example. Technical Report Research report OSD–T8402, Xerox PARC.

Hammond, K. J. (1989) *Case-based planning.* Academic Press.

Heise, R. (1989) Demonstration instead of programming: focussing attention in robot task acquisition. Master's thesis, Department of Computer Science, University of Calgary.

Heise, R. and MacDonald, B. A. (1989a) Robot program construction from examples. In *Proc. National Irish AI Conf.*, Dublin, Ireland, September. Also in press in book form, edited by Alan Smeaton, Springer Verlag.

Heise, R. and MacDonald, B. A. (1989b) Robots acquiring tasks from examples. In *Proc. ITESM 2nd Intl. Symp. on AI*, Monterrey, Mexico, October.

MacDonald, B. A. and Witten, I. H. (1987) Programming computer controlled systems by non-experts. In *Proc. IEEE Systems, Man and Cybernetics Annual Conf.*, pages 432–7, Alexandria, Virginia, October.

Maulsby, D. (1988) Inducing procedures interactively. Master's thesis, University of Calgary.

Maulsby, D., Witten, I. H., and Kittlitz, K. A. (1989) Metamouse: Specifying graphical procedures by example. In *Prof. SIGGRAPH'89.*

Michalski, R. S. and Chilausky, R. L. (1980) Learning by being told and learning from examples: an experimental comparison of two methods of knowledge in the context of developing an expert system for soybean disease diagnosis. *Int. Journal Policy Analysis and Information Systems*, 4(4).

Mitchell, T. M. (1982) Generalization as search. *Artificial Intelligence*, 18:203–206.

Myers, B. A. (1988) *Creating User Interfaces by Demonstration.* Academic Press.

Nix, R. (1983) *Editing by example.* PhD thesis, Computer Science Department, Yale University.

Nix, R. (1984) Editing by example. In *Proc. 11th ACM Symposium on Principles of Programming Languages*, pages 186–195.

Norman, D. A. (1980) Twelve issues for cognitive science. *Cognitive Science*, 4:1–32.

Pauli, D. (1990) Adaptive robot training. Explorations in sensorless manipulation. Master's thesis, Department of Computer Science, University of Calgary.

Segre, A. M. (1988) *Machine learning of robot assembly plans.* Kluwer Academic. Also Ph.D thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign.

Shaw, M. L. G. (1980) *On becoming a personal scientist.* Academic Press.

Shaw, M. L. G. and Gaines, B. R. (1989) Comparing conceptual structures: consensus, conflict, correspondence and contrast. *Knowledge Acquisition*, 1:341–363.

Takase, K., Paul, R., and Berg, E. (1981) A structured approach to robot programming and teachine. *IEEE Systems, Man and Cybernetics*, SMC-11(4), 274–289.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Opportunistic Acquisition of Plan Knowledge[1]

Dirk E. Mahling[2] and Oddmar A. Sandvik
Collaborative Systems Laboratory[3]
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Abstract

Motivation to enter expertise into a knowledge-based system is a major factor for the success of expert systems using knowledge acquisition modules. Another problem affecting the usability of expertise transfer systems is the ability to support groups of experts entering knowledge in a collaborative manner. By "disguising" knowledge-based systems under conventional interfaces, we can draw on prior skills and system acceptance of the users. Knowledge can be acquired en passant, *without the experts having to make a conscious effort. Traditional project management interfaces are used to illustrate this point. Information entered via accepted project management tools is organized into the underlying knowledge base formalism by using constraints offered by task and interface contexts. Facilities to coordinate expertise transfer efforts are based on shared knowledge structures, versioning mechanisms and consistency checkers. Project management functionality is augmented, using the knowledge base and inference processes.*

1 Introduction

Many workplace environments cannot allow experts to spend considerable time transferring their expertise into a knowledge-based system [BB89]. Experts are more likely to enter their knowledge into a computer when they can reap immediate benefits [CMN83, BS82], as exemplified in SALT [MMW85], spreadsheets [MFCS86], and existing project management applications. The immediate payoff from information entered satisfies the psychological need for closure [Lan87, HW83]. These motivational aspects have not received adequate attention in the area of knowledge acquisition systems [GJS88]. Developing strategies to acquire knowledge when experts are using conventional tools, such as project management systems, are necessary means to move knowledge acquisition systems from the laboratories into a wider range of applications. This approach recognizes that knowledge acquisition is not a separate activity, but takes place as the users work towards their professional objectives. The system should be able to acquire pieces of

[1]This work was supported by the Office of Naval Research under a University Initiative Research Grant, Contract # N00014-86-K-0764.

[2]Current address: German Research Center for AI (DFKI), Pf 2080, D-675 Kaiserslautern, Germany; E-mail: mahling@informatik.uni-kl.de.

[3]E-mail: sandvik@cs.umass.edu; Telephone: (413) 545-0609.

knowledge as the experts work on their normal job tasks, assemble scattered knowledge into coherent units, and give users the opportunity to utilize even incomplete knowledge bases.

Problems with current knowledge acquisition systems are also related to the experts' reluctance to learn how to use a new system with an unfamiliar interface and opaque functionality [Shn86]. Interfaces and systems requiring substantial training periods to support the job at hand are often disregarded by end users for slowing down the work process [Tho84]. The advantages usually gained from a knowledge-based system are far removed from the users' immediate concerns. This mental distance adds to the experts' reluctance to dedicate time to the expertise transfer process [HK85].

One approach towards solving this problem is the use of conventional interfaces, such as templates or spreadsheets, for knowledge acquisition. Experts are often familiar with a range of computer-based tools from their daily work environment. Project managers, for example, employ project management systems such as MacProject[4] to plan and track their efforts. Collecting information entered by experts using a traditional tool interface into an underlying knowledge base means expertise transfer *en passant*. The expert starts filling a knowledge base without a conscious effort. Not all pieces of a knowledge base can be acquired in this way, but they give the user an unexpected "added-value" by providing knowledge-based functionality not available in traditional tools, such as constraint checking, scenario experimentation, or plan elaboration. This "added-value" can serve as a motivational factor to enter missing parts of the knowledge base in a more traditional, acquisition-centered way. Very few systems use this approach towards knowledge acquisition. OPAL, for instance, provides data entry forms for therapy specification, which are very close to paper-based forms [MFCS86]. Still this is not the same as using existing and well known interface tools.

Furthermore, a single expert often does not possess knowledge covering the entire domain of expertise. Normally teams of experts cooperate on projects, with each person specializing in a particular area. To cover the domain, knowledge acquisition needs to be collaborative; multiple experts must work together to specify a complete knowledge base. This could either mean that a team of specialists enters pieces of knowledge synchronously, that a team cooperates on one piece of knowledge at a time, or that experts enter knowledge asynchronously. In addition to the acceptance and motivation problems mentioned above, coordinating the efforts of multiple experts requires a strong versioning system and facilities to direct others to particular areas in the knowledge base. Local and global consistency of the knowledge base must be maintained and users be allowed to resolve ambiguities quickly and reliably.

We have selected the project management domain as the starting point of our research efforts, using a project management support system as the front end of our expertise transfer system. Project managers are familiar with generic interfaces and diagrams in existing project management tools and theory. Furthermore, project managers are accustomed to providing descriptions of resources and people involved in projects, project activities, etc. In this paper we concentrate on the acquisition of activity knowledge.

[4]MacProject is a registered trademark of Apple Computer, Inc.

One way to categorize activities is based on the internal, human control of activities [Ras86b, Ras86a, FP62, Whi27]. On the lowest level we find skill-based activities that take place without conscious control. These activities are highly automated and integrated patterns of behavior. Tracing lines or simple assembly tasks fall in this category. In the next category, rule-based behavior, we find the conscious control of a sequence of skill-based behaviors. The control is stored in form of rules or procedures that can be acquired by trial and error, observation, communication, etc. Rules are matched to situations that contain elements fitting the rule. In unfamiliar or slightly different situations, however, rules cannot be applied. The control of performance must move to the next category of activities, goal-based behavior. In this category the goal is explicitly formulated based on an analysis of the situation and the general intentions of the person. Rule and skill-based behavior is subsumed by goal-based behavior. Different behaviors are tested mentally against the situation and the goal. At this level of functional reasoning, the person constructs a model of the situation and of the possible activities to perform.

Goal-based activity knowledge represents the most sophisticated form of activity knowledge in this categorization. It allows for the application of existing skills and rules in a constantly changing environment. More and more job positions require an increasing amount of goal-based activity knowledge. The knowledge needed to initiate, plan, execute and evaluate large projects can be viewed as the combined goal-based activity knowledge of its members. Following Rasmussen's hierarchy of activities, the scope of this project planning system is the acquisition of goal-based activity knowledge rather than skill-based or rule-based knowledge. The planning of activities in large projects is necessarily hierarchical and requires a large amount of reflection, backtracking, and adaptation to changing situations. In addition to activity knowledge it is also necessary to acquire knowledge about people involved in the project, their skills, consumable and non-consumable resources, project states, etc. Furthermore, the knowledge acquired is used to assist in project definition, hierarchical planning of projects, experimentation with different scenarios, automatic execution, and other unconventional, but highly desirable, functionalities for intelligent project management systems.

The next section introduces the underlying expert system architecture and knowledge representation formalism. Section 3 reviews some of the techniques and displays used in project management. The use of these displays for knowledge acquisition *en passant* is presented in section 4, while section 5 illustrates collaborative knowledge acquisition. Section 6 summarizes our opportunistic expertise transfer techniques and reports on the implementation status of the expert system and its expertise transfer component.

2 Underlying Architecture

The primary knowledge acquired in our project management settings is goal-based activity knowledge, also called plan knowledge. Figure 1 shows an instantiated example of the such an activity.

We derive the advanced project management functionalities and knowledge repre-

```
template-name:    ''Organize Symposium''
goal(s):          (status (symposium, staged))
decomposition:    (''Prepare Symposium''
                  (status (symposium, prepared)))
                  (''Conduct Symposium''
                  (status (symposium, conducted)))
                  (''Publish Proceedings''
                  (status (proceedings, published)))
precondition(s):(request (organization,symposium)),
                  (experienced (self, organizing))
side-effect(s): (gain (revenue, organization))
causal depend.: enables(prepare.symposium, conduct.symposium)
                  enables(prepare.symposium, publish.proceedings)
agent(s):         conference.organizer
constraint(s):  (sponsored.by(symposium, organization))
duration:         4 months
```

Figure 1: Activity (Plan) Description.

sentation schemes from POLYMER, a plan-based expert system [CL88]. POLYMER combines traditional project management tools with goal-based techniques and provides a consistent user interface to the project team. The user interface and traditional tools are based on commercially available systems such as MacProject. The interface additionally incorporates communication between multiple users and the advanced project management functionality provided by the plan-based expert system. POLYMER has been developed over the past three years as a testbed for goal-based support of cooperative tasks [CL88], providing a knowledge representation language for plans, activities, resources, and objects. It combines strategic and reactive planning, negotiation, backtracking, and plan exploration while interacting with multiple users. A goal-subgoal matching mechanism allows project representation, acquisition and display at different levels of abstraction, as well as flexibility to cope with a constantly changing environment. The main feature of hierarchical planning, posting a goal and searching the knowledge base for plans that can achieve that goal and are appropriate at the current state of the world, provides POLYMER with a powerful basis for supporting project managers. Figure 1 shows a POLYMER activity description in the project management application of symposium planning. Subgoals are captured in the decomposition-slot. At run time, POLYMER would try to find activities that have these subgoals as goals and satisfy the preconditions of the current state of affairs.

The architecture of POLYMER, adapted to the project management domain, is shown in figure 2. Users interact with the planner only indirectly, via a generic graph-

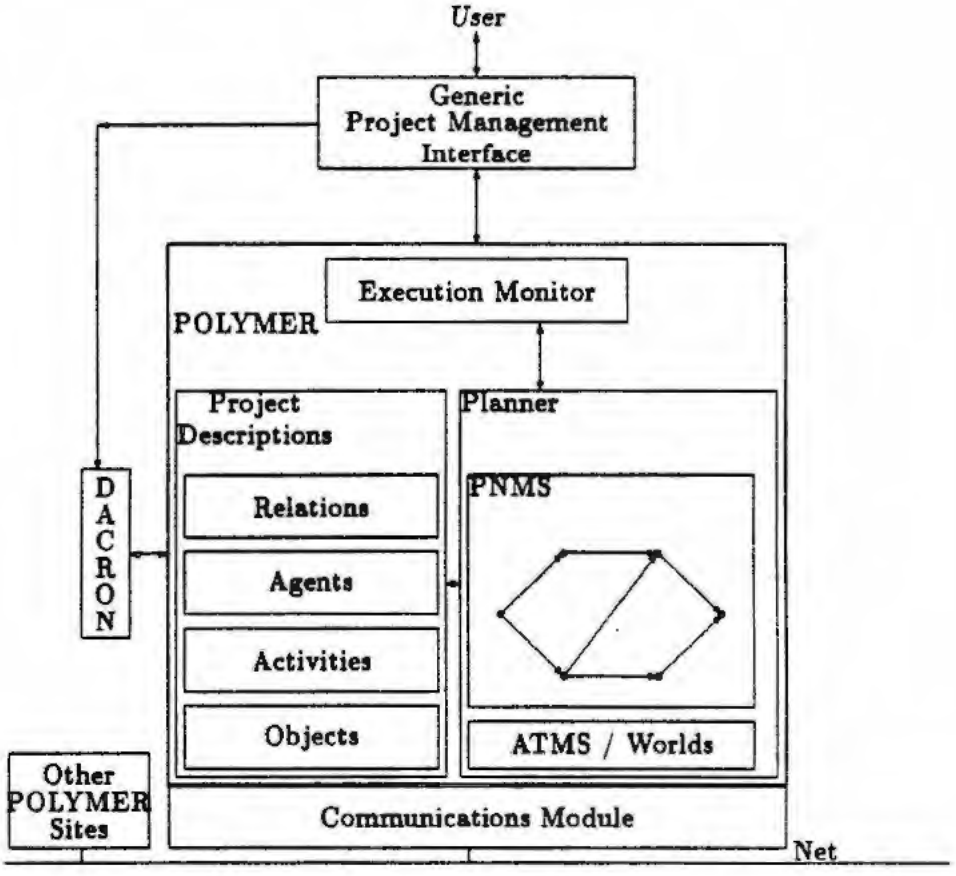


Figure 2: Distributed POLYMER Systems.

ical project management interface, similar to MacProject's interface. The DACRON system observes the project specification process and collects, assembles, and compiles pieces of knowledge into the domain model. The acquired knowledge is stored in POLYMER's knowledge base. Commands to the planner originate in the context of project management charts and diagrams and result in planner action. The planner creates plan networks which can be translated into the appropriate presentation languages. The result is planner interaction and display using familiar interface techniques, providing functionality far beyond that of conventional project management systems.

3 Project Management Techniques

One of our main concerns in aquiring project knowledge is to maintain continuity with interaction forms familiar to the expert. Not only does this address motivational issues, but it also forms a framework in which knowledge can be aquired *en passant*. A study was conducted to determine the needs and idiosyncracies of project management

[PS90]. This section presents some of the common knowledge representation and display formats preferred by project managers and indicates how these are applied in the project management system.

A Work Breakdown Structure (WBS) is an outline or task hierarchy diagram in which activities are specified in different levels of abstraction, or in increasing detail. The user interface provides a tree-structured WBS tool allowing users to specify tasks in such a manner, normally at the early planning stages where managers produce an initial list of activities comprising the project. Using familiar direct manipulation techniques, the manager can create activity icons at different levels of abstraction to incrementally refine the plan, specify temporal and causal relations among activities using arrows, and enter other task information.

PERT diagrams are used to indicate orderings among activities. In the project management system, the PERT interface can be used either to enter activities from a blank slate or to add temporal constraints by creating arrows between icons. A Gantt chart, on the other hand, displays activities in terms of durations and start and finish times. The user interface includes an interactive Gantt chart whereby users can also enter any task information or change the temporal relations between activities in a plan.

DACRON is a knowledge acquisition and display system that allows end users to enter descriptions of goal-based activities and generic objects [MC88]. It is based on a psychological model of human recall. Expertise transfer is achieved by copying icons and selecting from property menus. The main features are a visual language and editors for activities, local consistency checkers, and an animated display of plan networks using the visual language and graphical editors. Figure 3 shows a PERT diagram with its critical path, a WBS and a DACRON editor for the symposium planning domain.

4 Opportunistic Knowledge Acquisition

The knowledge acquisition paradigm advocated in this paper calls for incremental and opportunistic specification of knowledge base units. The expert will not have the time available nor the motivation to specify an entire plan in one session and the plan changes dynamically as the project is refined and developed. The system therefore allows for iterative plan refinement and accepts incomplete and underspecified knowledge to avoid artificial constraints upon the user. Knowledge is collected over many sessions into underlying units or "containers" represented in the POLYMER formalism (figure 1). Opportunistic knowledge acquisition allows the expert to augment the knowledge container when it is most natural in the work context. Furthermore, the system applies even the incomplete knowledge entered to provide immediate functional feedback or "added value", addressing the user's need for closure in knowledge acquisition.

The user interface tools (PERT, WBS, etc.) describe plan knowledge in a graphical language independent from the underlying plan formalism. These tools also implement a translation from a language project managers are familiar with to units in the knowledge base, hence expertise transfer *en passant*. Interacting with a tool that aids in plan specification and project monitoring, the expert will specify and refine information in

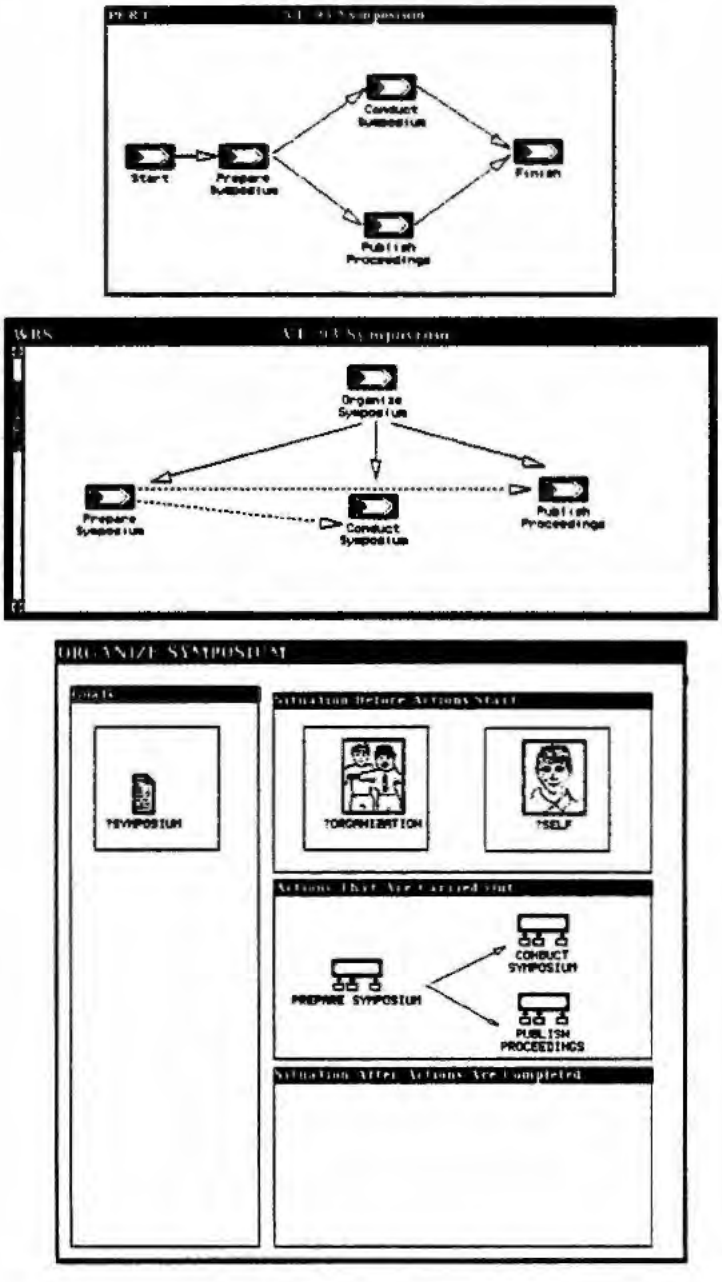


Figure 3: Same Task in PERT, WBS and DACRON.

the knowledge base while undertaking regular work tasks.

```
template-name:   ''Organize Symposium''
goal(s):         ()
decomposition:   (''Prepare Symposium''())
                 (''Conduct Symposium''())
                 (''Publish Proceedings''())
precondition(s):()
side-effect(s): ()
causal depend.: ()
agent(s):        ()
object(s):       ()
constraint(s):  ()
duration:        ()
```

```
template-name:   ''Prepare Symposium''
                 ...()
                 ...()

template-name:   ''Conduct Symposium''
                 ...()
                 ...()

template-name:   ''Publish Proceedings''
                  ...()
                  ...()
```

Figure 4: Activity Containers After First Step in WBS Tool.

Our scenario for illustrating how plan knowledge can be acquired and used is selected from the domain of symposium planning. Carol, a manager for XYZ Corp., has received a request from IEEE to organize VL-93, a symposium on visual languages. Carol decides to use the WBS tool to develop the initial project plan and creates an activity icon to represent **Organize Symposium** in the tree–structured interface. Using direct manipulation techniques, she adds three more icons for lower-level activities, thereby breaking the top level goal into three subgoals: **Prepare Symposium**, **Conduct Symposium**, and **Publish Proceedings**. Since these actions take place in the context of a WBS, the system creates containers or templates in the plan formalism corresponding to these icons with the appropriate labels. The subgoal decomposition of the top–level activity is induced from the tree and inserted into the decomposition slot of the **Organize Symposium** activity. Figure 3 shows the container for **Organize Symposium** created by these simple actions and the induced values of its slots, as well as the other three containers.

Carol decides to schedule the activities and switches to a PERT diagram in order to view the leaf nodes of the tree. She specifies temporal dependencies among the activi-

```
template-name:   ''Organize Symposium''
goal(s):         ()
decomposition:   (''Prepare Symposium''())
                 (''Conduct Symposium''())
                 (''Publish Proceedings())''
causal depend.:  enables(prepare.symposium,conduct.symposium)
                 enables(prepare.symposium,publish.proceedings)
```

Figure 5: Container after Insertion of Dependencies.

ties as arrows between icons (figure 3), corresponding representations are consequently created in the underlying formalism (figure 4). Carol can now proceed to add further constraints among activities or redefine existing ones by changing edges between nodes. This results in a change of the subgoal dependencies in the **Organize Symposium** container. Furthermore, Carol can specify preconditions and side-effects using dialog boxes with icons and pull-down menus. These are translated into the underlying representation and exemplify an instance of more directed knowledge acquisition. Carol knows from experience with the tool that adding such constraints increases the potential of the knowledge-based system to aid her in the project planning phase, as we will see below.

Now assume that Carol wants to achieve a higher level of detail in the WBS. POLYMER could help her to expand a leaf node if it had a formal description of the activity's goal. As Carol clicks on the WBS icon for **Prepare Symposium** and selects "expand" from the menu, the system informs her that a formal goal is needed and provides a DACRON editor to specify that goal. Figure 5 shows the graphical DACRON editor. Carol has copied icons from the graphical knowledge base into the goal compartment of this editor and specified a well formed formula for the goal in an iconic manner. Figure 6 shows the updated container resulting from these actions.

POLYMER attempts to match this goal with that of an existing plan to reuse previous project knowledge. If activities with corresponding goals are found, the system filters them by checking preconditions and other constraints and presents an appropriate plan to the user. In this example, POLYMER finds three activity descriptions with the goal **(status (symposium, prepared))**: **Prepare ACM Symposium**, **Prepare Small Symposium**, and **Prepare International Symposium**. The precondition of **Prepare ACM Symposium (sponsored.by (symposium, ACM))**, does not match the current state of the project. This leaves POLYMER with the other two activities, both of which are presented to Carol as DACRON editors. Carol selects **Prepare International Symposium**, because it is a large, official conference. The **Prepare Symposium** node in the WBS is expanded accordingly (figure 7).

Carol can make even more use of the knowledge entered in this and previous sessions. She can let POLYMER develop a whole project network for the task, complete with

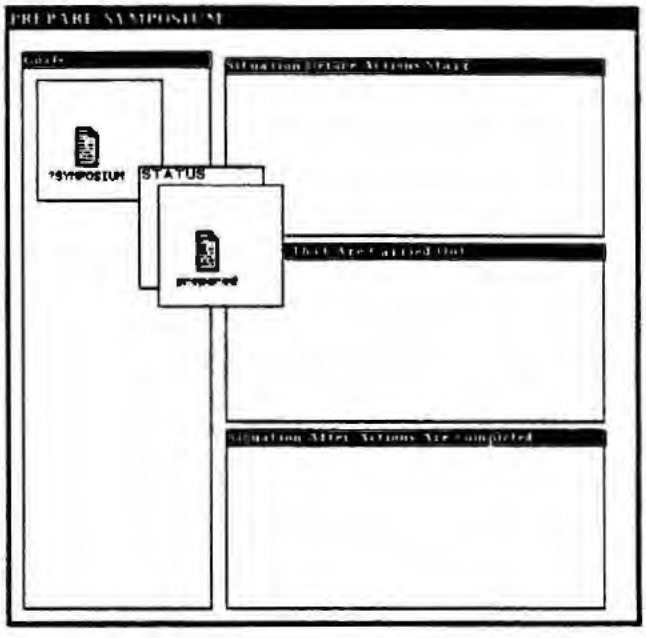


Figure 6: Iconic Goal in DACRON Editor.

value bindings, constraint checks and propagations, and subgoal expansions. In order to do that, Carol gives formal specifications for the three subgoals in her WBS, clicks on the **Prepare Symposium** node, and POLYMER starts expanding subgoals, effectively generating a plan network. This process ends when the leaves of the plan network are primitive actions that must be carried out either by a tool or by a person. Whenever Carol has choices of different activities to satisfy a subgoal or multiple legal values to bind to a varible, POLYMER presents Carol with choices in the form of icons or editors. Figure 8 shows the complete WBS for symposium planning, developed collaboratively by Carol and POLYMER.

5 Collaborative Knowledge Acquisition

Opportunistic knowledge acquisition is complemented by techniques that facilitate collaborative knowledge acquisition. While an expert may specify only parts of a knowledge unit *en passant*, such as a plan or a consumable resource, other experts may add parts to the same unit, thereby augmenting or finishing it. As shown in the previous section, even incomplete knowledge can be utilized by POLYMER to provide advanced functionality to the user. Augmentation of units can take the form of updating. Another form of collaboration in knowledge acquisition takes place when an expert specifies a plan that contains subgoals for which a colleague is responsible or has particular skills, in which case the system facilitates communication between the two experts to exchange specification requests. Mechanisms to guarantee knowledge base consistency alert experts to

```
template-name:   ''Organize Symposium''
goal(s):         ()
decomposition:   (''Prepare Symposium''())
                 (status (symposium, prepared)))
                 (''Conduct Symposium''())
                 (''Publish Proceedings''
                 (status (proceedings, published)))
causal depend.: enables(prepare.symposium,conduct.symposium)
                enables(prepare.symposium,publish.proceedings)
constraint(s):  (member(class (proceedings, PROCEEDINGS)))
```

Figure 7: Container with Formal Subgoal.

contradictions or ambiguities arising from data entered.

In our example, Carol provided only the name of the **Conduct Symposium** activity as she created icons in the WBS. Instances of plan templates are formed as result of that action (figure 3). The templates have the same name as the labels of the nodes in the WBS tree. Assume that Carol asks Bob to help her in scheduling the symposium. Using the PERT tool, Bob ascribes durations to nodes. These durations are also translated into the underlying plan representation. Figure 9 shows the updated containers.

The asynchronous, collaborative acquisition of knowledge uses the same principles employed by opportunistic knowledge acquisition techniques: creating containers corresponding to each knowledge unit that accommodate and organize incoming pieces of knowledge into the representational formalism.

Figure 3 shows the initial WBS created by Carol. She created four nodes graphically: **Organize Symposium**, **Prepare Symposium**, **Conduct Symposium**, and **Publish Proceedings**. As these actions took place in the context of a WBS, the four nodes result in the creation of four plan templates with corresponding names. The subgoal decomposition for the top level node (**Organize Symposium**) can immediately be induced from the tree. The scheduling expert Bob then obtains a PERT view of this WBS. The system translates the knowledge acquired from the plan formalism to a network of icons with links between them. Using this PERT representation, he finds that the temporal dependencies were not accurately entered in the WBS and corrects the PERT network by changing the edges between nodes. This results in a change of the subgoal dependencies in the formalism. As every cooperating expert accesses the same formalism, these changes will appear in the plans of every subsequent user. Carol, as the responsible project manager, will be alerted to any changes and may of course override them.

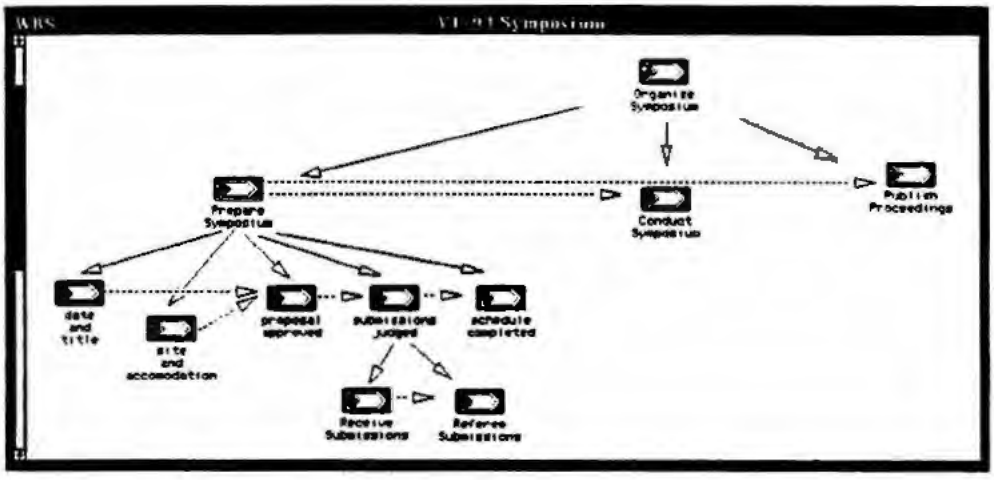


Figure 8: Expanded Work Breakdown Structure.

6 Summary and Status

We introduced the idea of acquiring knowledge via conventional interfaces, thereby reducing the experts' apprehensiveness to approach the system and increasing motivation by providing immediate use of information entered. We chose the project management domain as an example for this approach to expertise transfer. POLYMER, a plan-based expert system, was fitted with displays from the project management domain, such as PERT and Gantt charts and Work Breakdown Structures. We showed how each aspect of a POLYMER activity description could be acquired *en passant* during regular project management interactions. We also described how the acquired knowledge can improve traditional project management functionality.

POLYMER and DACRON have been completely implemented and tested. DACRON and POLYMER have been evaluated in the laboratory with more than thirty experts [Mah90]. Results show that experts are able to recall, specify, and use activity knowledge for projects. An interface library for PERT, Gantt, and Work Breakdown Structures exists. The complete interface system and its links to POLYMER are under implementation. A thorough evaluation with project managers is planned.

References

[BB89] J. Brule and A. Blount. *Knowledge Acquisition.* Mc Graw Hill, 1989.

[BS82] A. Badre and B. Shneiderman. *Directions in Human Computer Interaction.* Ablex, 1982.

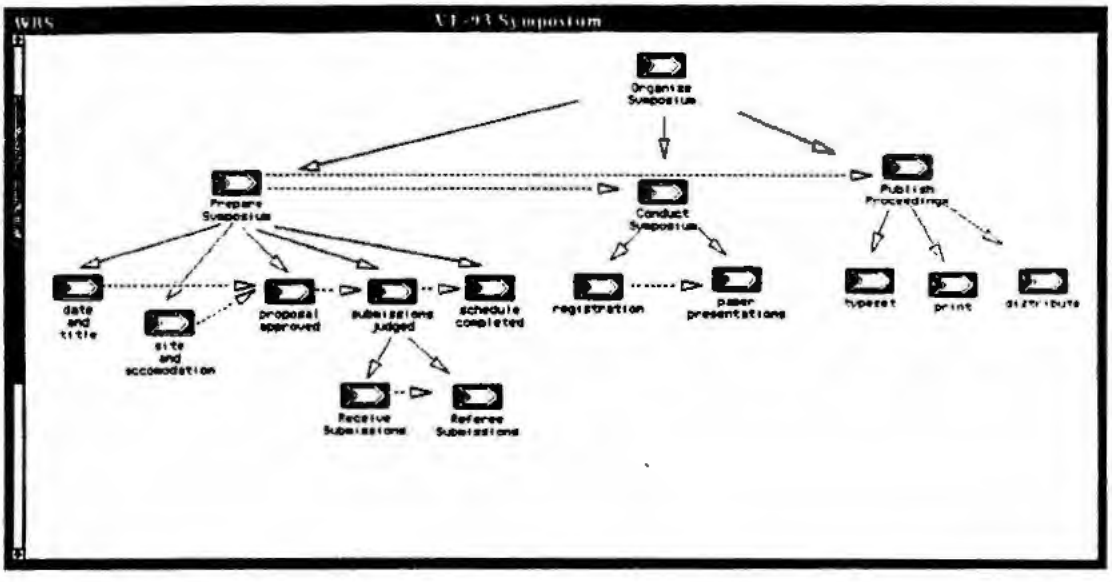


Figure 9: Complete WBS.

[CL88] W. B. Croft and L. S. Lefkowitz. Knowledge-based support of cooperative activities. In A. H. Bond and L. Gasser, editors, *Readings in Distributed Artificial Intelligence*, pages 599 –605. Morgan Kaufman, 1988.

[CMN83] S. Card, T. Moran, and A. Newell. *The Psychology of Human Computer Interaction.* Lawrence Erlbaum, 1983.

[FP62] P. M. Fitts and M.I. Posner. *Human Performance.* Brooks Cole Publ. Co., Monterey, CA., 1962.

[GJS88] C. Garg-Janardan and G. Salvendy. A conceptual framework for knowledge elicitation. In J. Boose and B. Gaines, editors, *Knowledge Acquisition Tools for Expert Systems.* Academic Press, 1988.

[HK85] H. Heckhausen and J. Kuhl. From wishes to action: The dead ends and short cuts on the long way to action. In J. Kuhl and J. Beckmann, editors, *Action Control: From Cognition to Behavior.* Springer-Verlag, 1985.

[HW83] E. Hollnagel and D.D. Woods. Cognitive engineering: New wine in new bottles. *International Journal of Man-Machine Studies*, 18(6), 1983.

[Lan87] T. K. Landauer. Relations between cognitive psychology and computer system design. In J. Carroll, editor, *Interfacing Thought.* MIT Press, 1987.

[Mah90] D. E. Mahling. *A Visual Language for Knowledge Acquisition, Display and Animation by Domain Experts and Novices.* PhD thesis, University of Massachusetts, February 1990.

[MC88] D. E. Mahling and W. B. Croft. Knowledge acquisition for planners. In B. Gaines and J. Boose, editors, *Proceedings of the 3rd Knowledge Acquisition for Knowledge Based Systems Workshop*, 1988.

[MFCS86] M. Musen, L. Fagan, D. Coombs, and E. Shortliffe. Using a domain model to drive an interactive knowledge editing tool. In *Proceedings of the Knowledge Acquisition for Knowledge Based Systems Workshop*, pages 33.0–33.11. American Association for Artificial Intelligence, 1986.

[MMW85] S.J. Marcus, J. McDermott, and T. Wang. Knowledge acquistion for constructive systems. In *Proceedings of the ninth international conference on artificial intelligence*, Los Angeles, California, 1985.

[PS90] C. Pietras and O. Sandvik. A review of project management tasks and tools. Technical Report CSL-90-1, University of Massachusetts, Collaborative Systems Laboratory, 1990.

[Ras86a] J. Rasmussen. A framework for the cognitive task analysis in systems design. In E. Hollnagel, G. Mancini, and D.D. Woods, editors, *Intelligent decision support in process environments*, pages 175–210. Springer Verlag, 1986.

[Ras86b] J. Rasmussen. *Information Processing and Human-Machine Interaction.* Systems Science and Engineering. North-Holland, 1986.

[Shn86] B. Shneiderman. *Designing the User Interface: Strategies for Human Computer Interaction.* Addison Wesley, 1986.

[Tho84] J. Thomas, editor. *Human Factors in Computer Systems.* Ablex, 1984.

[Whi27] A. N. Whitehead. *Symbolism, its meaning and effect.* Macmillan, New York, 1927.

```
template-name:  ''Organize Symposium''
goal(s):
decomposition:  (''Prepare Symposium''
                (status (symposium, prepared))
                (''Conduct Symposium''())
                (''Publish Proceedings''())
precondition(s):
side-effect(s):
causal depend.: enables(prepare.symposium,conduct.symposium)
                enables(prepare.symposium,publish.proceedings)
agent(s):
object(s):
constraint(s):
duration:       4 months
```

```
template-name:  ''Publish Proceedings''
                ...
duration:       1 month
                ...
```

```
template-name:  ''Conduct Symposium''
                ...
duration:       5 days
                ...
```

```
template-name:  ''Prepare International Symposium''
goal(s):        (status (symposium, prepared))
decomposition:  (''date and title''),(''site and accomodation'')
                (''proposal approved''),(''submissions judged'')
                (''schedule completed'')
precondition(s):(citizen (participants, international))
side-effect(s):
(gain (revenue, organization))
causal depend.: ()
agent(s):       ()
object(s):      ()
constraint(s):  ()
duration:       3 months
```

Figure 10: Activity Containers after PERT Operations.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

KATE: A Piece of Computer Aided Knowledge Engineering

Michel MANAGO

CNRS & *IntelliSoft*
16 Passage Foubert
75013, Paris, FRANCE

Yves KODRATOFF

CNRS & équipe inférence et apprentissage
Laboratoire de Recherche en Informatique, bât 490
Université de Paris sud, 91405, ORSAY, FRANCE

abstract : We present KATE, a knowledge acquisition system which integrates inductive learning techniques (i.e. learning from examples) and a frame based representation language (structured objects). KATE can use background domain knowledge and does not solely rely on observed statistical regularities. The frame-based language is used to represent hierarchies of classes and descriptors (taxonomies), constraints, and procedural calls which enable deducing descriptors from other descriptors (goal-driven or data-driven reasoning). The system is designed around the ID_3 architecture but it can handle complex databases of examples with objects, relations between objects, variables and it can use background domain knowledge. It also offers additional tools for knowledge acquisition such as an object editor, a spreadsheet to enter the examples, graphical browsers, a decision tree editor, a runtime generator, external interfaces to databases and programming languages etc. KATE also provides methodologies for knowledge acquisition, knowledge base validation and expert system maintenance. It is a commercial product which has been validated on several real life applications.

1. *Turning Data into Knowledge*

Since the early 1980's, induction tools have been used to generate knowledge based systems from databases. From a set of training cases, an induction system automatically builds a knowledge base in the form of a decision tree or a set of production rules. For instance, from a database of patients whose diseases are known, the induction engine learns diagnostic rules that are then used to identify the disease of new incoming patients.

The ID_3 algorithm (Quinlan, 1983) is such an induction system. Its descendants have been used for numerous real-life, large scale, industrial applications (Michie, 1989). Nevertheless, all the potential applications for knowledge-based systems technology cannot be tackled by ID_3. It has been our experience that its knowledge representation capabilities are too limited to cope with the training data when it is made out of complex entities and that it lacks the ability to handle objects and relations. Due to the increasing sophistication of Database Management Systems (relational and objet-oriented DBMS), there is a clear gap between what can be achieved with ID_3 and the needs to fulfil. This was our motivation for developing an induction tool based on the ID_3 algorithm (same search techniques, same preference criterion) but which has more powerful knowledge representation language. The scope of applications which can be built using this induction technique is thus wider.

2. *Learning from Complex Data*

In this section, we discuss why an attribute-value language such as ID_3's is not appropriate for some real life application that have been tackled using KATE. Consider the domain of tomato plant diseases. A diseased tomato plant can be affected by several different symptoms. Each symptom is a complex entity, or object, whose description depends on its type and on its location. A symptom of type spot is not described the same way as a symptom of type hole (ex: a spot has a color, a hole does not). Likewise, a symptom on a leaf is not described the same way as a symptom on a fruit (the position of the symptom with respect to the nerves of the leaf is relevant for describing a symptom on a leaf but not for describing a symptom on a fruit). For instance, the symptom *tache-ou-plage-sur-folioles* (spot-on-leaves) is described by the following 17 features:

> REPARTITION-SUR-PLANTE, REPARTITION-SUR-FEUILLE, NOMBRE, COULEUR, ZONATIONS, MESURE, LIMITES, LOCALISATION-SUR-FOLIOLE, PROPORTION-SUR-FOLIOLE, REPARTITION-SUR-FOLIOLE, JAUNISSEMENT-POURTOUR, MYCELIUM-FRUCTIFICATIONS, ASPECT-OU-MYCELIUM, TOUCHER, FORME, RELIEF.*

Another symptom, *chancre-extérieur-collet* (canker-on-the-outside-of-the-stem), is described by the 14 features:

> ZONATIONS, TOUCHER, MESURE, FORME, LIMITES, RELIEF, LIEN-SYMPTOME-INTERNE, DEGRE-D-ATTAQUE, MYCELIUM-FRUCTIFICATIONS, REPARTITION-SUR-PLANTE, LOCALISATION-SUR-EXTERIEUR-TIGE, REPARTITION-SUR-EXTERIEUR-TIGE, NOMBRE, COULEUR**

These two complex objects (symptoms) share some common features such as COULEUR (color) and FORME (shape). They also have some unshared features: JAUNISSEMENT-POURTOUR (yellowing-around-the-edges) for the spot on leaves, LOCALISATION-SUR-EXTERIEUR-TIGE. (location-on-the-outside-of-the-stem) for the canker on stem etc. In the tomato application, there are **147** different kinds of symptoms. On the average, each kind of symptom is described by 9 features. Furthermore, more than one symptom (out of these 147) can appear in the description of a training example (there are up to 6 symptoms for the same example). How can we represent such an example with a flat attribute-value vector like the one used in ID_3?

One possibility is to introduce an attribute for each type of symptom (ex: exists-spot-on-leaves) and an attribute for each of its features (ex: color-of-spot-on-leaves). Unfortunately, since there are 147 different symptoms with an average of 9 features per symptom, this yields a total of 147 * 9 = 1323 attributes. Such a large number of attributes generates so many information gain computations that ID_3 fails to find a solution (combinatorial explosion). At the root node, ID_3 makes 1323 information gain computations, then for each son 1322 computations and so on. In this application, the search space is thus very large because each example is complex to describe. In addition,

---

* distribution-on-plant, distribution-on-leaf, number, color, spherical-zones, measure, limits, localization-on-folioles, proportion-on-folioles, distribution-on-folioles, yellowing-of-edges, mycellium-fructifications, aspect-of-mycelium, feel, form, scraps

** spherical-zones, feel, measure, limits, scraps, link-to-internal-symptom, state-of-damages, fructification-of-mycelliums, distribution-on-plant, localization-on-the-outside-of-the-stem, distribution-on-the-outside-of-the-stem, number, color

with this choice of representation, although it is possible to have several different symptoms in the description of the same example, it is impossible to have two or more symptoms of the same type. For instance, one cannot adequatly represent and example where there are two symptoms of type spots-on-leaves with different colors and shapes.

There is a second solution for modelling the data using an attribute-value vector which is more economical in terms of the total number of attributes required. For the previous solution, when a feature is shared by two different symptoms, two attributes are generated so that there can be more than one symptom in the description of the same example. For instance, canker-on-the-outside-of-the-stem and spot-on-leaves share the color feature but, with the above choice of representation, the two attributes color-of-spot-on-leaves and color-of-canker-on-stem are generated. We could thus take advantage of the fact that the symptoms have some features in common in order to reduce the number of attributes. A special attribute called "type-of-symptom", which has one of 147 values (the 147 kinds of symptoms), is introduced. We then take the union of all the features for all the attributes and associate an attribute to each one. This yields a total number of 80 attributes per symptom. Thus, since there is an average of 9 features per symptom, 71 attributes are, on the average, marked as irrelevant. For example, color is not relevant for describing a hole-in-leaves but it must still appear in the description of this symptom since it is useful for describing some other symptom such as a spot-on-leaves. Since the same tomato plant (i.e. an example) can have up to 6 different symptoms, these figures are multiplied by 6. Indeed, if there is at least one example with 6 symptoms, we must introduce 6 time as many attributes in order to describe all the examples. This yields a total number of 480 attributes for describing the symptoms of an example, 429 of which are irrelevant.

As we have seen, in both cases we end up with quite a large number of attributes to describe each example. Unfortunately, ID_3 is good at handling databases with a large number of examples, but the examples are usually described by few attributes (less than 100). Since the mathematical complexity of ID_3 is linear with the number of examples, and in $O(n^2)$ with the number of bits of information needed to represent an example (Manago, 1988), having a very large number of attributes is a source of problems.

In addition, we must introduce a special value to mark that an attribute is not meaningful in a certain context. For example, if we choose the first representation, we must introduce a special value for the attributes of the symptoms which do not exist (i.e. what is the value of color-of-spot-on-leaves in an example where exists-spot-on-leaves = no?). If we choose the second representation, we must mark that a feature is irrelevant for describing a certain kind of symptom (i.e. what is the value of color when the symptom is hole-in-leaves?). This value, which we will call "irrelevant", is processed by the induction engine in a special way so that it does not disrupt the information gain computation: the training examples with value "irrelevant" are propagated in each branch of the node.

Unfortunately, in our application, this simple approach produces such a large number of computations that the induction system fails to complete its task. Let us assume we have chosen the second, more economical, solution for representing the data. Consider a symptom represented by a flat vector of attributes such as the one presented below.

	TYPE-SYMPTOME-1	COULEUR -1	...	JAUNISSEMENT-POURTOUR-1	REPARTITION EXTERIEUR-TIGE-1
EX109	TACHE-OU-PLAGE-SUR-FOLIOLES	BRUN	...	OUI	IRRELEVANT
EX206	CHANCRE-EXTÉRIEUR-COLLET	GRIS-BEIGE	...	IRRELEVANT	GÉNÉRALISÉE

The set of attributes which describe the symptom is the union of all the features for all the possible symptoms (80 attributes). On the average, there are 71 attributes with value "irrelevant".Therefore, during induction, as soon as the type of the symptom is known, ID_3 makes **71** useless computations of information gain for that symptom **at each subsequent node** of the decision tree. Since there can be up to 6 symptoms per plant, there are 6 times as many attributes and ID_3 makes 6*71=429 totally useless information gain computations at each node.

Furthermore, the number of features needed for a symptom is not the only number that is drastically increased due to the fact that ID_3 does not use common-sense knowledge about the objects it classifies. The set of possible values for these features is also increased. For instance, the set of possible colors for a symptom of type TACHE-OU-PLAGE-SUR-FOLIOLES is different from the one of a symptom of type CHANCRE-EXTÉRIEUR-COLLET. Using ID_3's attribute-value representation, the set of legal values for each feature is the union of the set of values possible for the feature when it is attached to any symptom. The legal values for the color attribute include the ones for a symptom of type TACHE-OU-PLAGE-SUR-FOLIOLES and for a symptom of type CHANCRE-EXTÉRIEUR-COLLET. In the tomato domain application, this yields an average of 30 different possible values for each attribute. Since in its computation of information gain, ID_3 evaluates the entropy at each branch (i.e. each possible value for the attribute), ID_3 makes 429 * 30 = **12870 useless computations of entropy** for irrelevant features plus a large number of entropy computations for irrelevant feature values for the remaining relevant features **at each node** of the tree.

As we have seen, due to the fact that ID_3 does not use common sense knowledge about the entities which describe the examples, it can perform a massive amount of useless computations. This problem also arises in other applications such as in the cancer database (Quinlan, 1986) where a patient is described by its sex and its state of pregnancy (the feature "pregnant" is not relevant in all examples where "sex = male"). However, in this last application a patient is described by 30 attributes and the computational overhead turns out to be marginal. On the other hand, it cannot be afforded when the training data is complex and when the number of irrelevant features becomes overwheelming like in the tomato diseases application (combinatorial explosion during induction). In this case, background knowledge about the domain must be used in order to constrain the search space and remove irrelevant features: the induction engine should not even compute information gain of being pregnant for patients who are known to be males!

There are other fundamental reasons why ID_3 cannot be used for this application. For example, it cannot represent adequatly relations between objects. These may turn out to be discriminant (i.e. the fact that the mold is on the spot is important). The point we are trying to make is that the reason why ID_3 cannot be used in this application is not because its induction algorithm is inadequate, but because its knowledge representation formalism is insufficient. Having noticed this simple fact, the obvious idea is to build an induction tool with the same learning mechanism as ID_3 (hill-climbing search strategy, heuristic preference criterion based on entropy) but with a more powerful knowledge

representation language. This idea of integrating the two sounds quite simple, but we had to overcome considerable technical difficulties in order to achieve this goal.

3. *Constraining Search with Frames during Induction*

3.1. The Frame-Based Language in KATE

For several years, frame-based knowledge representation languages (schemas, units or object-oriented languages) have invaded most of the industrial Artificial Intelligence tools. The practical interest of these knowledge representation languages has been well demonstrated : they are powerful, the knowledge is made of modular reusable entities, the formalism is natural and it enables object-oriented programming. In this section we briefly present KATE's frame-based language to show how we represent background domain knowledge to cope with the problems described in the previous section.

A frame can be viewed as a data structure. It represents a set of object (a *class)* or of the objects themselves (the *instances* of the class). Properties and relations, called *slots*, are attached to frames. KATE makes a distinction between the private slots of the class (called *own slots)* that are not transfered to the instances and the one that are transfered to the instances of the class (called *member slots)*. During induction, only member slots of the class are used for building the tree. Facets are attached to the slots. This enables representing different informations about the slots. For example, there are facets that represent the actual value of the slot ("value" facet), the type of values expected for the slot ("type" facet), procedures that can deduce features from other features (daemon facets

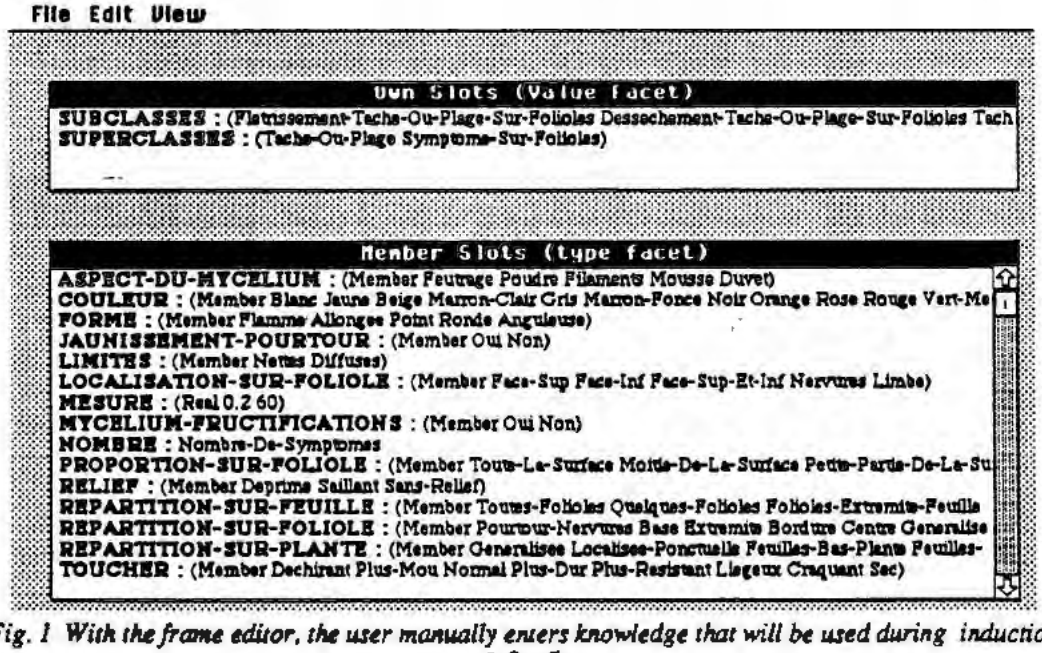


Fig. 1 With the frame editor, the user manually enters knowledge that will be used during induction

such as methods and active-values), the number of values that are possible for a slot ("cardinal" facet) etc. Consider the following frame :

```
[TACHE-OU-PLAGE-SUR-FOLIOLES
own slots :
        <SUPERCLASSES (VALUE TACHE-OU-PLAGE SYMPTOMES-SUR-FOLIOLES)> etc...
member slots :
        <LOCALISATION-SUR-FOLIOLE
                (TYPE (MEMBER FACE-SUP FACE-INF NERVURES LIMBE)
                 (CARDINAL (1 2))>
        <MESURE
                (TYPE (REAL 0.2 60))>    etc.]
```

The "type" facet of member slot LOCALISATION-SUR-FOLIOLE indicates the range of possible values for this nominal attribute. The value of this slot is one of the values in the set {FACE-SUP FACE-INF NERVURES LIMBE}. The "cardinal" facet attached to this same slot indicates that the slot can have between 1 and 2 values. Thus, in some of the examples this feature will have one value and in some other it will have two values (in the above set of possible values). The "type" facet attached to the MESURE member slot indicates that the slot expects a real value between 0.2 and 60. In KATE, predefined datatypes include nominals (the value is in a set of discrete values), ordered nominals (the value is in a list of ordered values), integers (or subranges), reals (or subranges), booleans or user defined classes. In the last case, the slot is a relation that links an instance of a class to another instance of a class.

Frames may be organized in a hierarchy of generality as shown in figure 2. The most general concepts appear toward the roots of the hierarchy. For example, the value facet of own slot "SUPERCLASSES" indicates that the concept of spot-on-leaves inherits slots from its two superconcepts spot and symptom-on-leaves. For instance, LOCALISATION-SUR-FOLIOLE has been inherited from SYMPTOMES-SUR-FOLIOLES and MESURE from TACHE-OU-PLAGE. This hierarchy is entered manually using the frame editor and is part of the background knowledge. Note that one does not have to provide these hierarchies in order to use KATE. However, if these are available as input, KATE is able to use them.

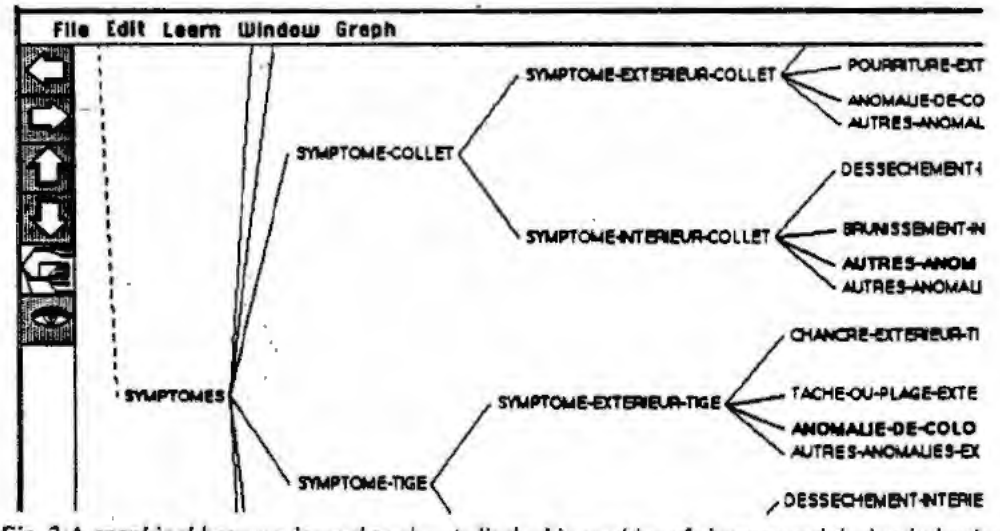


Fig. 2 A graphical browser is used to show/edit the hierarchies of classes used during induction

Multiple inheritance reduces the initial biases introduced by the hierarchy. The hierarchy of symptoms can be organized both according to the location of the symptoms or according to their nature. Without multiple inheritance, the user would have to choose one or the other, and this choice would have an impact on the behavior of the learning system.

3.2. *Constraining the Set of Relevant Features and Feature Values during Induction*

By using information represented with the frame-based language, KATE knows which features are relevant for describing an object. The induction engine of KATE uses this knowledge when building the decision tree. This is achieved by always introducing a feature with the type of the object on which it is attached. For subsequent nodes in the decision tree, the features that are possible are constrained because of the type of the object. For example, if at the first node of the decision tree the most informative feature has been the SIZE slot of a symptom of type HOLE-IN-A-LEAF, further down in the tree, the slots attached to the frame HOLE-IN-A-LEAF and only these are used for the information gain computation. Descriptors such as COLOR, TEXTURE etc... that are not relevant for this object are simply ignored. Thus, at different nodes of the decision tree, different features are considered. The search strategy (hill-climbing) and preference criterion (maximize information gain) used in KATE is similar to the ones used in ID_3. The difference lies in what is considered during the search. While ID_3 considers all the remaining features which do not already appear in the tree for the computation of information gain, KATE dynamically constrains the set of features which are considered.

Note that the computational cost of testing wether a slot belongs to a class or not, as it is done in KATE, frame is fixed while the cost of computing information gain of an attribute, as it is done in ID_3, grows linearly with the number of examples. Moreover, this cost is marginal: it is simply the cost of accessing the slots of a class[1]. Thus, by constraining the set of attributes which are relevant for an object we have not replaced some computations by some other and we have not lost anything in terms of efficiency of the induction algorithm.

Information gain is thus computed for a feature that is attached to an object of a certain type. For example, KATE tests the information gain of COLOR(SYMPTOM-ON-LEAVES) and not the information gain of COLOR by itself as ID_3 would. As we have seen in the previous section, the information about the type of the slot (expected values of the slot) is represented by a facet, called type, that is attached to the slot. Thus, two different frames may have different type facets and KATE is able to represent the fact that the range of values for a slot is different for different frames. Since the range of values for a slot when it is attached to a frame is known, and since information gain is computed for a slot

---

[1] Inheritance does not affect the computational cost. In our implementation of KATE, a list of candidate nodes is maintained by adding or removing features when a candidate node is choseen as the current decision node. During induction, we never traverse the whole frame hierarchy to retrieve the relevant features of an object. The computational cost for constraining the set of relevant features is thus the cost of accessing the local slots of a class frame and not the cost of inheritance.

attached to a certain object, KATE is able to constrain the set of relevant values for that slot : each branch for the candidate node in the tree is meaningful in the current context.

As we have seen, the frame-based language is used to represent common sense knowledge about the entities which describe the examples. This knowledge is then used during induction to constrain the set of features that are relevant in a given context (i.e. reduce the number of test nodes that are considered for the information gain computation) and to constrain the set of values for these features (i.e. reduce the number of branches for these test nodes). As we have seen in section 2, the set of irrelevant features can rapidly become overwhelming and prevent the induction system from finding a solution (combinatorial explosion). Note that any dataset which can be represented by ID_3's feature-value vectors can also be trivially represented using objects. One simply introduce a single class frame (i.e. the examples are represented by a single object) whose slots corresponds to the ID_3 features. KATE then behaves exactly like a ID_3. However, for datasets that cannot be adequatly modelled by feature-value vectors (variable number of objects, relational predicates/slots etc.) KATE's induction engine can still be used.

3.3. Handling Examples with a Variable Number of Objects

The procedure which dynamically generates candidate nodes in KATE for the information gain computation is described below.

- When all the examples at the current branch of the decision tree contain an object of the same type, information gain of the slots of this object is computed and the one with highest information gain is selected as the current node
- When all the examples do not contain an object of the same type, information gain of the test "Exists-object(object-type)" is computed.

This enables dealing with examples that are described by different objects. For instance, when all the examples contain a symptom of type SYMPTOM-ON-LEAVES, KATE computes information gain only for the features that describe an instance of the class SYMPTOM-ON-LEAVES (and of its descendants in the frame hierarchy). Once the first symptom appears in the tree, information gain of the remaining relevant features of the symptom is computed. Information gain of introducing a second symptom and of specializing the type of the first symptom is also computed. The system never considers features attached to a second symptom until one symptom already appears in the tree, features attached to a third symptom until two symptoms already appear in the tree etc...

For the humans database (Quinlan, 1983), the result of introducing a ninth fictitious example of class "-" which does not contain an object of type humans is shown in figure 3. Note that we used this well known dataset to illustrate the technique but, of course, having an example without a human is totally meaningless. However, consider a database of families where the examples have more than a single object of type humans. What if there are couples as well as bachelors in the data? The test exist(humans) becomes interesting as soon as the first human already appears in the decision tree. In the tomato plant dataset, the situation is similar since a single diseased tomato plant is described by a

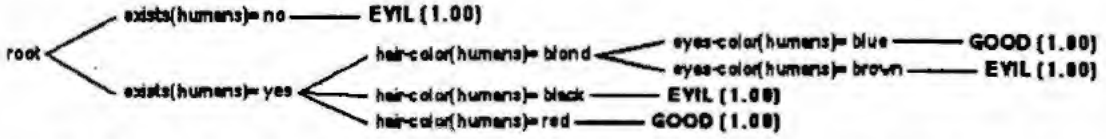


Fig. 3 When the training examples contain a variable number of objects, information gain of "Exists(object-type)" is considered when building the tree

variable number of symptoms (between 1 and 6). We are thus interested in computing the information gain of having a single symptom versus having two symptoms etc.

Note that the fact that there is more than one object of the same type in the training examples would lead an induction engine based on propositional calculus (such as ID_3) into learning incorrect results even if it could handle the combinatorics. To represent the fact that there can be 6 symptoms per examples with a feature-value vector, one would have to introduce 6 features called exists-symptom1, ..., exists-symptom6 and several other features to describe each symptom (such as color-symptom1,..., color-symptom6, shape-of-symptom1,..., shape-of-symptom6). Nevertheless, the choice of the label "symptom1"..."symptom6" for a given symptom is purely random. There are no valid reasons to decide that a particular spot-on-folioles should be called "symptom1" instead of "symptom2". Consider what might happen if in every examples of a disease "A", there is a spot-on-folioles called symptom1. ID_3 might then learn a rule such as "IF <type-of-symptom1 = spot-on-folioles> THEN disease A". However, if there is a "spot-on-folioles" that is called "symptom2" in an example of disease "B" this rule is semantically incorrect. In KATE, the first test "exist(object-type)" that appears in the decision tree means "is there any object of that type in the example?", the second test means "is there a second object of that type in the example?" etc. Each object that appear in the tree is given a name that can be refered to further down in the tree. This identifier is a logical variable that is bound to actual objects in the training examples so that we can refer to these particular objects when computing information gain of other features afterward.

4. *Object Oriented Programming in KATE*

The object-orientation of KATE is a unifying factor for all its individual tools. Not only does it enable modelling the domain entities and the training examples using an object oriented approach, but it also enables object-oriented programming. Object-oriented programming can be very useful for deducing knowledge (goal-driven or data-driven reasoning), for implementing the tools (ex: the spreadsheet is implemented in KATE's object language) or for interfacing the different tools one another or with the external world. In this section we describe how the induction engine is integrated in the toolbox and how it interacts with the other modules in order to provide a complete knowledge acquisition plateform.

4.1. *Using Deductive Knowledge*

Procedural calls (daemons) are used to deduce features from other features. For example, the values of slots "type-of-culture" and "heating-mode" of frame "cultures" can be deduced from the slot "kind-of-culture". When "kind-of-culture" is "in-the-fields", "type-of-culture" is "in-ground" (in a glass-house it could be sand) and "heating-mode" is "cold" (in a plastic tunnel or a glass-house, it could be heated). The backward chaining rule "IF <kind-of-culture=in-the-field> THEN <heating-mode=cold>" can be efficiently implemented by a method attached to slots "type-of-culture" and "heating-mode". The following method is attached to the "heating-mode" slot of frame "culture". Message is a primitive of the KATE programming language and *self* is a system variable bound to the original recipient of the message, as it is classically done in object-oriented programming. The "$" prefix denotes an entity in the current knowledge base.

```
IF (EQUAL (message *self* $kind-of-culture) $in-the-field) THEN $cold
```

In addition to KATE's object language, the user has access to external programming languages such as LISP, Pascal and C and he can use the full power of a real programming language.

KATE uses deductive knowledge in several tools. In the spreadsheet, used to collect the database of training examples, deductive knowledge saves the work of filling information that can be automatically deduced, as shown in fig. 4.

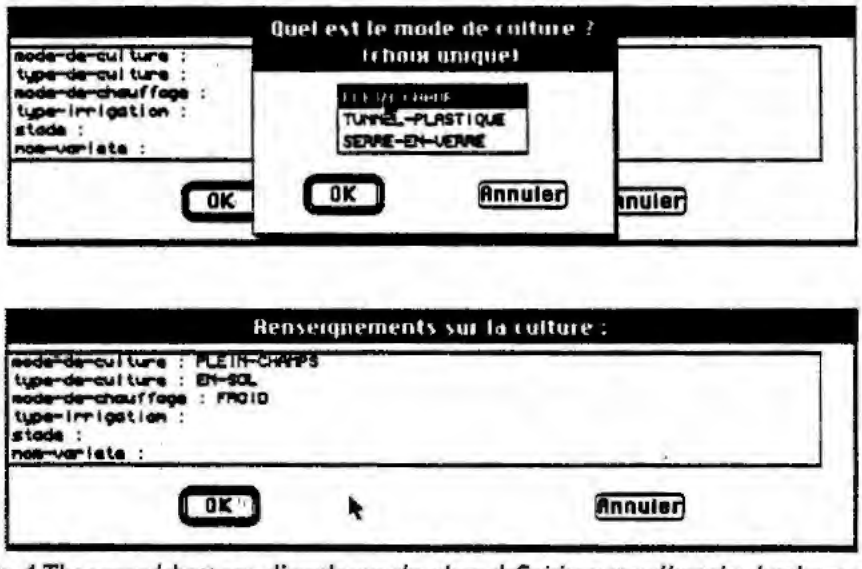


Fig. 4 The spreadsheet can directly use the class definitions to collect the database of examples

The values of slots "type-of-culture" and "heating-mode" are automatically filled in when the user enters the value of "mode-de-culture". This deduction is performed by the methods attached to slots "type-de-culture" and "mode-de-chauffage" of frame "culture" (goal driven reasoning). The methods are activated when accessing the unknown value of these slots. The methods dynamically compute the values of these slots and deduce that,

when the slot "kind-of-culture" has value "in-the-fields", its value is "cold". Note that if the value of "kind-of-culture" was itself computed dynamically by another method, we would have a backward chaining deduction.

Deduction can also be achieved in forward chaining (event driven) by attaching an active value daemon to the slot. For example, one could attach an active value to slot "kind-of-culture" which would update the values of "heating mode" and "type-of-culture".

Should the induction engine also use background deductive knowledge? This issue is highly controversial (i.e. is it better to empirically discover everything from scratch, or to use deductive domain knowledge when it has been provided?). One of the advantages of the knowledge intensive approach is that incorrect data may be filtered by using domain knowledge as shown in (Manago & Kodratoff, 1987). One of its disavantages is that it introduce yet another bias on the behavior of the learning system. We feel that it is rather difficult to state that one way is better than the other and that this is very much application dependant: for some applications it is better to use deductive knowledge during induction, for some it is better to use pure induction. In KATE, we have chosen compile the training examples and saturate their description prior to induction. During compilation, all the slot values which can be deduced are filled in. The user has the option to specify wether a value that is deduced has a lower or a greater priority than the value which was entered manually (in case of conflicts) and thus he can choose wether deduction has a higher precedence than empirical discovery.

4.2. Interface to External Databases

Object-oriented programming is not only useful for deducing features from other features. It can be also used to connect KATE to external devices that automatically fill up the values of the slots and to interface KATE with an external database. Object oriented programming is also used to implement a spreadsheet to collect and view the database of training examples. With KATE's spreadsheet, one can build an on-line interactive questionnaire that can be customized: the spreadsheet's reactions to a mouse click can be locally redefined by using a programming language or by linking the spreasheet to Hypercard through an eval server.

Object-oriented programming has been one of the key factor for interfacing expert system shells of the market to databases. Several leading shells offer an object-oriented front end for databases access. The schema of the database maps into the objects and the data is linked to the expert system shell in a manner that is completly transparent for the user.

KATE offers an interface to database management systems and external spreadsheets through ASCII files. An abstract description of the file's content is provided in the class definitions by attaching a position-in-datafile facet to the slots. Using this information, KATE is able to read the training examples directly from the datafile. Each record is separated by carriage returns in the datafile and each component of a record is separated by tabulations or blank spaces. In Fig. 5, we show the class description of a datafile for lending money to companies. This particular datafile was generated using MicroSoft® Multiplan™ on a PC and contains 700 records which were all used for induction.

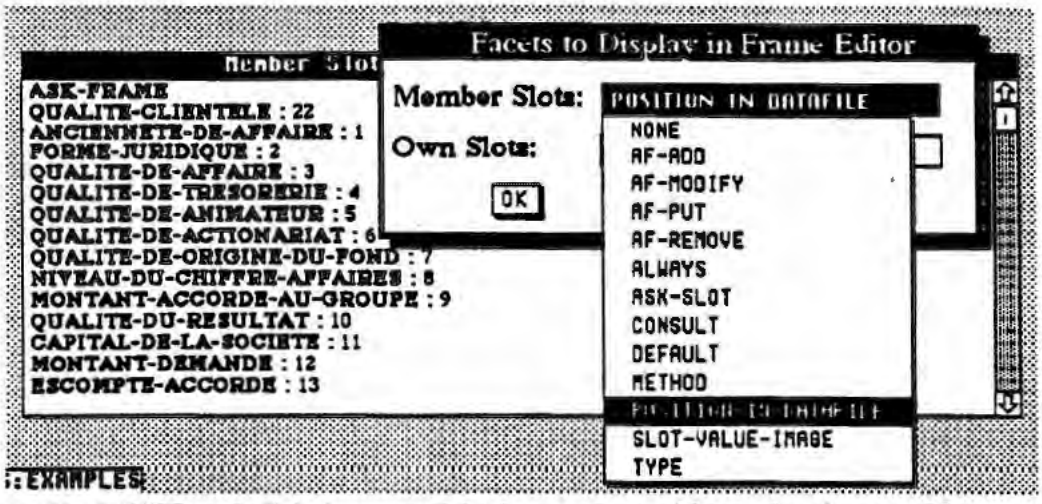


Fig. 5 KATE can be linked to external databases and spreadsheets using classes and facets. The value of the facet is the position of the slot value within the record in the datafile

1	95	SARL	S	S	?	?	B	P	?	J
100	?	?	S	?	S	?	?	?	95	?
?	?	?	?	?	?	S	J	?		

Fig 6. A data record for a company asking for a bank loan. Question marks denote unknown values.

With this simple interface, the object structure can virtually be linked to almost any database. Thus, if a database of cases already exists, the knowledge acquisition step is reduced to saving the data in an ASCII file and defining the structure of the domain entities with the frame editor. KATE is then able to directly import this datafile.

4.3. The Object-oriented Spreadsheet

Often, induction tools are only used when there is already a database of training cases available. However, even when there is no such database available, one can still benefit from using induction during the knowledge acquisition stage because induction tools offer methodologies for :

- knowledge acquisition
- knowledge base validation
- expert system maintenance

The human expert does not have to model his domain with abstract rules which he may not be familiar with. Indeed, Knowledge Acquisition (KA) is not merely the process of transfering a mental model that lies somewhere within the brain of a human expert, but the familiar scientific and engineering problem of formalizing a domain for the first time. One of the major drawbacks of the classical method for knowledge acquisition, called *knowledge elicitation*, is that it relies on the expert's ability to explain his own reasoning. Unfortunately, human experts are trained at solving practical problems, not at explaining how they solve them. Using induction, the expert always works on actual cases. He does

not have to explain how he solve the problem, he simply validates the examples (i.e. provide the class of the example). The induction tool is then responsible for learning a decision tree which derives the same conclusions as the expert.

KATE can be used for applications for which there is no database available. It offers a spreadsheet that can be used to generate the database of examples. Typically, the lines of the spreadsheet correspond to the slots of the objects. The behavior of the spreadsheet when the user clicks in one of the slots is implementted by messages (object-oriented programming). We describe below the standard implementations of the messages:

- When the user clicks in a nominal slot (i.e. a slot whose value is in a list of values) a menu is displayed so that he can choose the value of the slot for the current example (see fig. 4). Symbolic links between different slots can be made with object-oriented programming. In fig 6, a link is made between "mode-de-culture" and "mode-de-chauffage". As we have seen, the value of the second slot can be computed automatically from the value of the first one. Since it is computed by a method in backward chaining, if the user changes his minds and modify the value of "mode-de-culture", the value of "mode-de-chauffage" is also updated. The two are symbolically linked like the cells of a classical spreadsheet. A graphical representation of the slot can also be used instead of a text representation by attaching a facet with the image file's name to the slot.
- When the user clicks in a numerical slot (fig. 7), a dialog box appears so that he can type the value of the slot. A check is made to ensure that the value falls in the allowed range.
- When the user clicks in a relational slots, the user is asked through a menu to give

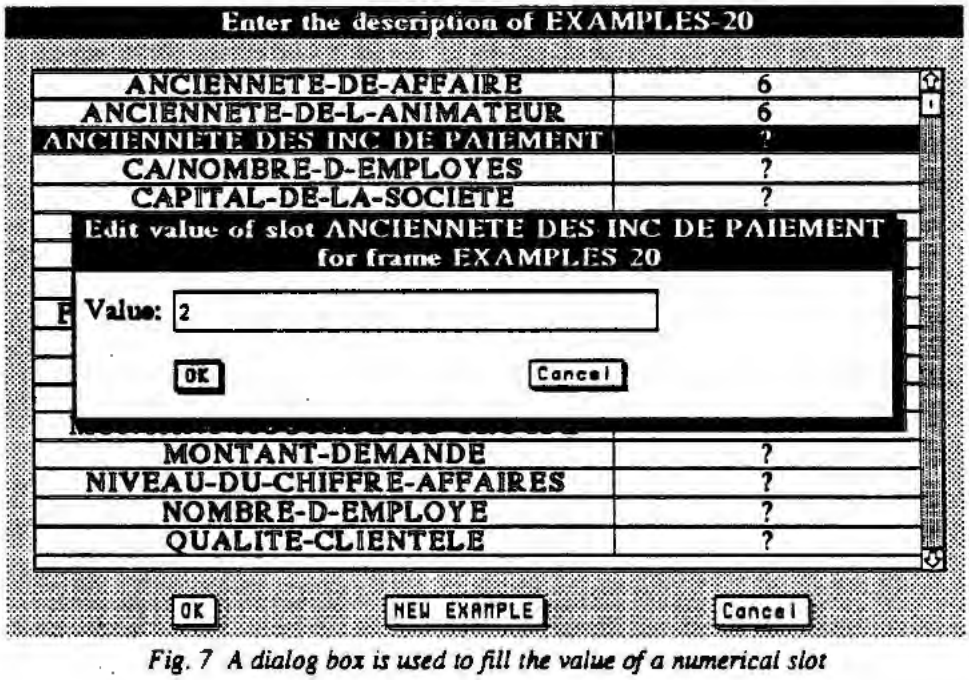


Fig. 7 A dialog box is used to fill the value of a numerical slot

the class of the object that the current object is linked. An instance of that class is then created and the user is taken to this object. In general, there is a sheet for each object and the user easily navigates between sheets with the mouse.

The advantage of object-oriented programming for implementing the spreadsheet is that the spreadsheet can be customized afterward. The procedure for filling an object (ask-object message) can be locally redefined for a certain class of objects, or slightly modified by adding before and after daemons for the class. The procedure for filling the value of a slot (ask-slot daemon) can be customized for a given slot of a given class. There is also a bridge between the spreadsheet and Hypercard (fig. 8) to customize the interface, and to programming languages (Lisp, C and Pascal) so that the cells can be automatically filled in by a computer program which retrieves their values from sensors.

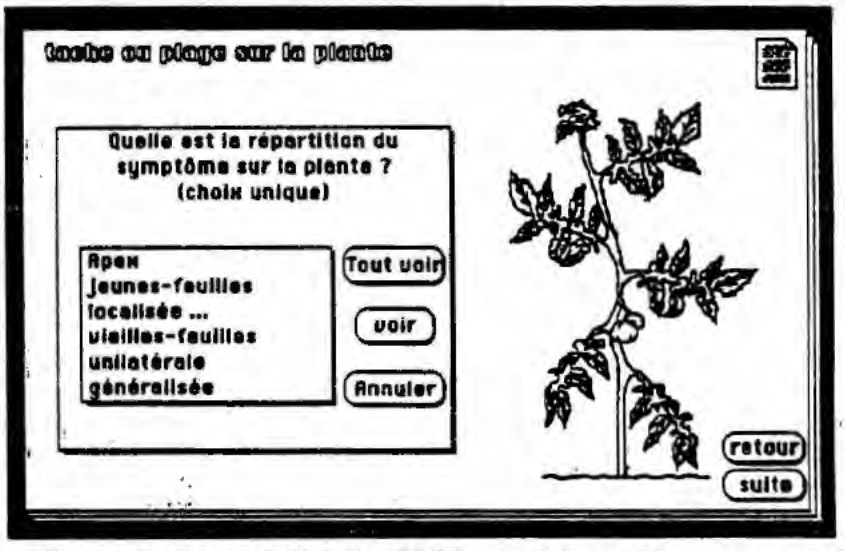


Fig. 8 A Hypercard stack can be linked to KATE's spreadsheet and interactive consultation

The spreasheet is not only used as an interactive questionnaire. It also allows to simulate training cases. A study is currently being conducted at Renault Automation to develop a fault diagnostic expert system for a new robot on an assembly line. Since the robot does not yet exists, one cannot provide a database of actual faults for that robot. Using the spreadsheet, AMDEC grids can be entered. The AMDEC method was originally used in the space and airplane industry to prevent risks due to component failures. It is used to improve security and reduce maintenance cost when designing new robots. The data in the grids is quite complex and some basic understanding of how the machine works (encoded using an object-based model of the domain) is needed to build, through simulation, a database of examples and generate a meaningful decision tree. In this application, all the information contained in the grids is not always interesting for the induction engine. The AMDEC grid is basically is a large faults-consequences graph from which the signatures of the faults can be extracted (i.e. the error message on the video, the LED which is lighted on the control panel etc.) and generate a case library of faults. Using the simulated cases, a decision tree which identifies the faults can be produced.

5. KATE : A Knowledge Engineering Tool

KATE is a toolbox which assists the knowledge engineer in rapidly building applications. The human expert always has the last word even when his own solution is not considered by KATE as being optimal. In the decision tree editor, the user can select a node and list all the test with their information gain. He has the freedom of changing the test which appear at that node and replace it with one that he prefers. The induction engine can be restarted from this node or he can keep on editing the tree by hand. Induction is thus only viewed as a useful and practical knowledge acquisition technique and nothing more. KATE does not solely rely on induction to generate a knowledge base and the user can always feed into the system his own expert knowledge.

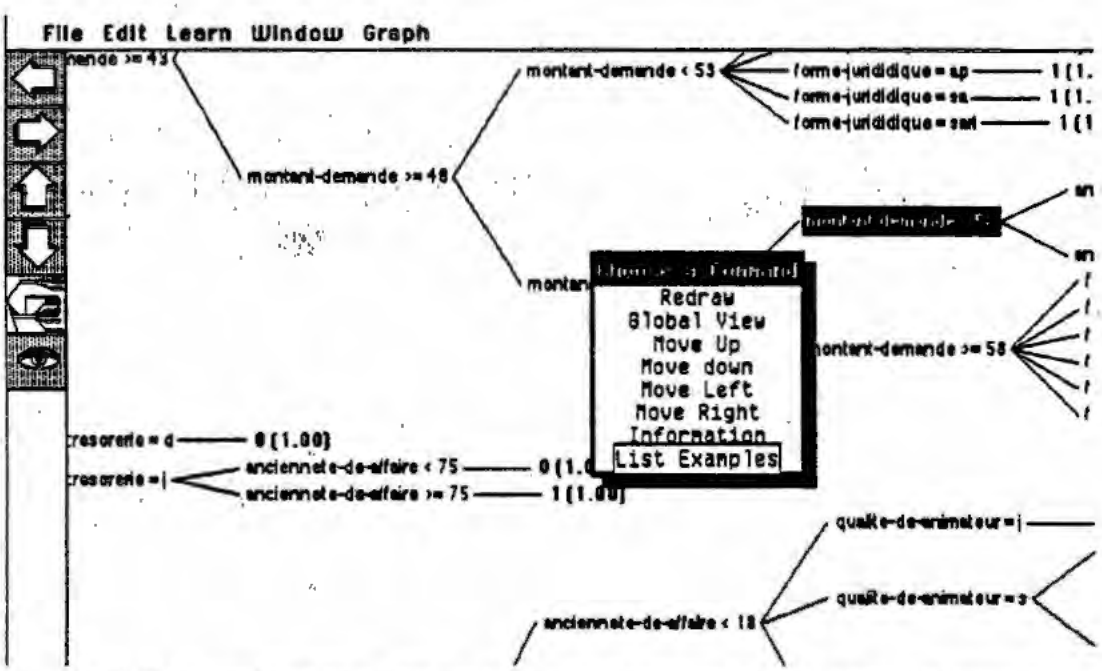


Fig. 9 The examples below a given node can be easily accessed through a graphical browser

Several tools are available so that the user can evaluate the learnt knowledge. In the decision tree browser (fig. 9) the user can obtain the list of examples below a given node so that he can get a better understanding of why the induction tool chose a particular test. In addition to easily browsing and inspecting the decision tree, he can also generate a list of declarative rules which are simpler to understand for large applications (decision trees can become bushy and are difficult to understand). The rules are generated from the tree but have fewer preconditions since the decision tree structure is pruned and the numerical tests are collapsed into single tests (for instance, A < 10 & A < 15 & A < 22 is collapsed into A < 10). The rule pruning algorithm we use is given in (Corlet, 1983). After looking at the result of induction, the expert can find holes in the knowledge which has been entered into KATE. For instance, some examples belonging to different concept can be covered by the leaf node in which case a probabilites are attached to the concepts. This shows that either there is some true uncertainty in the domain, or that the expert forgot to

provide some important infomation which he could access. He then goes back to the frame editor to modify the domain theory (i.e. enter new slots to describe the examples). Using default value facets and the spreadsheet, he then updates the description of the training cases, goes back to the induction engine and iterate this process. KATE can thus be used to rapidly prototype applications: enter a partially accurate model of the domain, enter some training cases, generate a tree, evaluate the learnt knowledge and change the domain model or the examples. Induction is integrated in a knowledge acquisition cycle instead of being simply considered as a one-shot data-analysis tool.

6. *Applications*

KATE is more specifically aimed at applications such as :

- maintenance of industrial machines
- design and validation of decision systems
- computer-aided generation of diagnostic expert systems
- expert systems maintenance
- analysis, interpretation and classification of complex data
- quality control in manufacturing

We must insist on the fact that induction is not only useful for knowledge acquisition but also for knowledge base validation and expert system maintenace as it is shown below.

6.1. Diagnosis in Plant Pathology

KATE has been used to generate, and maintain an expert system on tomato plant diseases at the Institut National de Recherche en Agronomie (INRA). Note that some promising preliminary results on generating a diagnostic system for tomato plant diseases had been achieved throughout the course the INSTIL ESPRIT project (INSTIL, 1989). In the tomato application, there is a small number of examples per concept (there are less than 400 examples for 60 diseases). There are even some diseases for which there is no example since, according to the expert, it takes about 7 years to see all the diseases. For this application, it is therefore meaningless to use a training set and a test set to measure the accuracy of the learnt rules as it is usually done in machine learning research and extrapolate results which are not significant. The learnt rules have been evaluated empirically by the human expert. One of his conclusions was that the rules describing features of existing objects seemed to be meaningful. In fact, the first time the system ran on the data, it found the exact rule he had given for the disease of "Moëlle noire". However, he did not like negative rules such as the one shown below:

IF exists(tache-ou-plage-sur-foliole) = no & exists(anomalie-de-la-forme-ou-de-la-taille-sur-foliole) = no & exists(jaunissement-sur-foliole) = no & exists(autres-anomalies-sur-foliole) = no & exists(jaunissement/dessechement-sur-foliole) = no & exists(dessechement/tache-ou-plage-sur-foliole) = no & exists(fletrissement/jaunissement-sur-foliole) = no & exists(ravageurs-sur-foliole) = no & exists(tache-ou-plage/autres-anomalies-de-coloration-sur-foliole) = no & exists(jaunissement/tache-ou-plage-sur-foliole) = no & exists(fletrissement/tache-ou-plage-sur-

foliole) = no & exists(fletrissement-sur-foliole) = no & exists(dessechement-sur-foliole) = no & exists(autres-anomalies-de-coloration-sur-foliole) = no THEN Oidium (0.20), Pvy (0.80)

This rule says "if there are none of these 16 symptoms, the disease is "Oidium" with a probability of 0.2 or "Pvy" with a probability of 0.8. In this application domain, considering that each year there variations in existing diseases (mutation, changes in the climate etc.) and even new diseases that arise, the expert simply refuses to view his application domain as a closed world. Although a purely negative rule is consistent with the training data, it is totally meaningless for him.

The final conclusion is that, for this application, induction is not so useful for knowledge acquisition considering the difficulty of obtaining the training examples. However, it is extremely interesting for expert system maintenance. Since in tomato plant pathology, it is not unusual that a new disease appear or that an existing disease mutate and presents some unseen features, the ability to maintain rapidly the knowledge base from one crop season to the other is of vital importance. As a consequence, the decision tree that is generated by KATE is not used for interactive consultation as it is usually done (i.e. to each node of the tree corresponds a question during consultation). Instead, the consultation procedure is based on the on-line interactive questionnaire (i.e. built using the object-based spreadsheet) which is normally used to collect the full description of a training example. The user then chooses to record the description as a training example (the expert provides his diagnosis afterward) or to consult the decision tree through KATE's auto-consultation. The expert system is then maintained by regenerating the decision tree as new examples are integrated in the database. INRA is currently evaluating this methodology for expert system maintenance for other applications. Apple France is also interested in this methodology for building fault diagnosis systems which will be made available to their vendors via the French Numeris (RNIS) telecommunication system (through Apple's EasyLink® protocol).

6.2. Credit Assesments

The french "Société d'Informatique et de Système (SIS)" has tested KATE on an application delivered to one of France's major bank (Compagnie Bancaire). The expert system's task is to evaluate financial risks when the bank covers credit assessed by a supplier to one of its customer. SIS is a large french servicing company with 20 years of experience in business data processing and 6 years of experience in expert system technology. They have designed an in-house expert system shell devoted to building applications in business data processing (SIS-EXPERT) and have used induction before (Expert Ease). They are also using classical statistical analysis techniques (the SAS system) and they have conducted a thorough comparison of knowledge acquisition with KATE, knowledge elicitation by interviewing the human experts and knowledge acquisition using statistical analysis for this commercial application.

The training set presented to KATE contains 700 examples described by one object with 31 slots: 12 slots with integer values, 2 slots with real value, 17 slots with nominal values (with an average of 5 values per slot). There are 7 concepts to identify. The problem is complicated because on the average 80% of an example's description is unknown. Considering the massive amount of unknown values, one cannot associate to the value a

probability that is computed on the rest of the training set as it is described in (Quinlan, 1989). When building the tree, KATE propagates examples with unknown values in each branch of a decision node. This drastically increases the complexity of the algorithm (i.e. there are over 10000 examples at the leaves of the decision tree since the same examples can appear in more than one leaf). Furthermore, the bank has to make a quick decision for a very large number of cases and the information available is mostly qualitative (ex: a parameter such as "quality of the company's manager" is rather subjective). The constrain is that the decision taken has to be explained and justified to the customer. The rule that is extracted from the decision tree provides this justification.

14 men days were required to build the system from scratch using KATE from an existing database. KATE was ran three times and the background knowledge was slightly modified in between each run. For instance, in between the first two runs, the expert added some useful ratios such as the total revenue of the company divided by the number of employees. A method was attached to the slot to automatically compute this ratio from the information available. The resulting tree was tested using KATE's auto-consultation mode on a test set of 400 unseen examples. KATE made 70,3% perfect predictions (same response as the expert) and an additional 21,7% correct predictions (the right answer was suggested with the highest probability). Note that 5% of the examples in the training set were classified with uncertainty through auto-consultation which demonstrates that the human experts had access to some additional information not available to KATE. The following table summarizes the results of the comparison between the panel of experts, KATE, the statistical analysis tool SAS and knowledge elicitation (interviews of experts).

	board of experts	KATE	knowledge elicitation	statistical analysis
perfect (%)	91	70.3	73.2	48.6
good (%)	8.1	21.7	12.3	26.8
errors (%)	0.9	8	14.5	24.6
time required		14	60	25
man power		11	45	15
analyst		8	7	9
experts		1	8	2
specialist		2	30	4

"Perfect" means the exact answer that was expected, "good" means that the correct answer was suggested with a probability above 0.5, "errors" means that the system answered something incorrect, failed to answer or suggested the correct response with a probability that was too low, "time" means the total time it took to build the system, "man power" means the total man power that was required to build the application. The man power has been broken in between the specialist of the method (knowledge engineers for knowledge elicitation, specialists of the method for induction and statistical analysis), human experts who have been interviewed or who have evaluated the learnt knowledge base and computer analysts. The figures have been produced and published by the customer in a comparative evaluation of the techniques for building financial applications (Perray, 1990). Note that, for this application, the frame structure of KATE is not useful. The data is represented by a single object whose slots correspond to ID_3's attributes. This proves our claim that any applications that can be tackled using ID_3 can also be tackled using an induction tool with a more elaborated representation language such as KATE.

6.3. *Military Applications*

KATE has also been used for generating and validating a fault diagnosis expert system (errors of transmission in a circuit for the US army) and a military decision support system (aid for interpretation of 2-D pictures) at the Centre d'Electronique de l'Armement de Rennes (this last application is classified). Let us simply state that, like for the tomato application, the training data is represented by complex objects and that a flat attribute-based representation would not have adequatly captured the information contained in the training data. A second military application for identification of equipment (land vehicles) is being built at the CELAR. The system currently identifies one family of military equipment (tanks). The tank database consists of 134 examples of combat tanks which belong to 14 different nationalities. There are currently 12 classes of objects for describing a tank and a total of 58 slots (8 relations, 9 integers, 41 nominals). Again, the description of an example varies depending on the type of the objects involved (for instance, a tracked tank is not described in the same way as a tank with wheels). There are currently no hierarchies of classes in the knowledge base but this will change shortly when the other families of vehicles will be entered. In the whole application, there are 12 different families to identify (mostly vehicles) with roughly 100 examples per family. Note that this is a single application, as opposed to 12 different ones, since the conclusions can overlap when there are unknown features during consultation. For example, due to camouflage, tanks and artillery might look similar.

Conclusions

In this paper, we have presented a tool that integrates well known concepts in machine learning, software engineering and knowledge representation so as to expand the scope of applications that can be tackled using induction. KATE can represent complex training cases that could not be handled by first generation induction tools based on propositional calculus such as ID_3. What have we lost with respect to ID_3? Nothing, because ID_3 is a special case of KATE. When the training data is represented using a single object, the slots of the object correspond to the attributes in ID_3 and the dynamic node generation procedure is only called once at the start of induction. However, when the training data is made of complex entities, KATE can still be used while ID_3 cannot. In addition to providing an up-to-date knowledge representation language, KATE offers several tools which makes it a complete knowledge acquisition plateform: a frame editor to graphically enter the initial domain model (features to describe the examples, deductive knowledge etc.), a database interface to import existing training cases when they are available, an object-based spreadsheet to build the database of examples, graphical browsers to view the class hierarchies and the decision tree, a production rule generator to evaluate the learnt knowledge, a decision tree editor to inspect and modify the learnt knowledge, interfaces to Hypercard and to programming languages, the KATIOUCHA runtime generator which allows delivering on micro computers applications that were built with KATE etc. KATE's object-orientation is a unifying factor for the different modules and the induction engine is fully integrated in this knowledge acquisition toolbox.

Acknowledgements

We would like to thank *IntelliSoft* for allowing us to publish this work, Noël Conruyt, Jacques Lerenard and Dominique Blancard (INRA) who have developped the tomato diagnosis application. *IntelliSoft* would also like to thank all the organisms that support machine learning research at *IntelliSoft* : the European Economic Community (ESPRIT project 2154, Machine Learning Toolbox), l'Agence Nationale pour la Valorisation de la Recherche (ANVAR), le Ministère de la Recherche et de la Technologie (MRT) and Apple computers. KATE™ is a trademark of *IntelliSoft* inc. and is available on the microExplorer™ from Texas Instruments. KATIOUCHA™ (KATE's runtime generator) is available on the Macintosh®.

References

Corlett, R. (1983). "Explaining induced decision trees," *in Proc. Expert Systems, pp. 136-142.*

INSTIL (1989). Project Summary. ESPRIT Deliverable. Brussel, Belgium: Commission of Economic Communities.

Manago, M. & Kodratoff, Y. (1987). "Noise and Knowledge Acquisition," *in Proc. of the 10th International Joint Conference in Artificial Intelligence,* pp. 348-354, Morgan-Kauffman.

Manago, M. (1988). Intégration de Techniques Numériques et Symboliques en Apprentissage Automatique. PhD dissertation, University of Orsay, France.

Manago, M. (1989). "Knowledge Intensive Induction," in *Proc. of the sixth international workshop on machine learning,* Segre A. M. ed, pp. 151-155, Morgan-Kaufmann.

Manago, M. & Kodratoff, Y. (1989), "Induction of Decision Trees from Complex Structured Data," *Proc of the IJCAI -1989 workshop on Knowledge Discovery in Databases,* Piatesky-Shapiro, G., Frawley, W. eds, pp. 219-229. To be republished in a book by AAAI Press in 1990.

Michie, D. (1989). "New Commercial Opportunities Using Information Technology", in Proc. 3rd International Gi-Kongress, W. Brauer, C. Freska eds, pp. 64-71, Springer Verlag.

Perray, M. (1990). Etude Comparative Entre Trois Techniques d'Acquisition des Connaissances: Interview, Induction et Analyse Statistique pour Construire une Même Base de Connaissances. *Proceedings. of the Journées Informatique et Intelligence Artificielle.* Paris, France.

Quinlan, J. R. (1983). "Learning efficient classification procedures and their application to chess end games," in Machine Learning 1: An Artificial Intelligence Approach, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann.

Quinlan, J. R. (1986). "Simplifying decision trees," *in Proc. of the 1st AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems,* pp. 36.0 - 36.15.

Quinlan, J. R. (1989). "Unknown Attribute Values in Induction," in *Proc. of the sixth international workshop on machine learning,* Segre, A. M. ed, pp. 164-168, Morgan-Kaufmann.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Interactive Acquisition of Design Decisions

William Mark
Jon Schlossberg
Lockheed AI Center
3251 Hanover St. 96-20 B/259
Palo Alto, CA 94304
(415)354-5236 Mark@sumex.stanford.edu

Abstract

In almost all design efforts, previous design experience provides guidance: offering comparisons, warning of pitfalls, and sometimes furnishing components or structure for the new design. The use of experience is usually implicit in the design process, based on designers' ability to retrieve and determine the relevancy of previous design efforts (most often only their own). We believe that design productivity could be greatly improved if designers had easy access to an on-line memory of designs. We discuss a technique for interactively accessing and "growing" a memory of design decisions.

1 Introduction

Design is a process of decision making. The designer makes decisions about structure, component choice, etc., at various levels of detail. Constraints change, compromises are reached, and so on until a result is obtained (or the effort is abandoned). The design process is labor-intensive, with high value-added; it is certainly cost effective to use "lessons learned" in future design efforts.

In fact, almost all design efforts do make use of previous design experience to provide comparisons, warn of pitfalls, and sometimes furnish components or structures for the new design. But the use of this experience is usually limited to individual designers' ability to retrieve and determine the relevancy of previous design efforts (most often only their own).

This is not due to a lack of well documented designs. Almost all organizations have records – on paper or in CAD systems – of the detailed specifications for and results of their design efforts. The problem is that it is very difficult to see the relevance of these records for new design efforts, since they do not record the decisions that led to the final design, the alternatives that were considered and the rationale for not selecting them, and so on.

"Design experience" is therefore usually limited to what one or a small group of designers can recall of past similar efforts. This recall is valuable precisely because it consists of memories of *design processes*, including the relevant decisions and their rationale.

2 Design Memory

We believe that design productivity can be greatly improved by giving designers access to an on-line memory of designs that incorporates the decisions and rationale that make up the designs. Such a facility would serve as a resource for recording design processes within a company or even within an industry; as a vehicle for information sharing among the various participants in a large design project (to enable "concurrent engineering" or "design for manufacturability"); or as an institutional memory for a system that is designed to have a long lifetime.

This paper sketches an approach for building a design memory. The basic idea is to involve designers in an interactive memory-based design process that results not only in improvement of design productivity, but also in cumulative growth of the memory itself.

Designs are represented in terms of the decisions they comprise. As designers create a new design, they make decisions that can be compared with decisions made on similar designs in the past. Designs intended to implement similar functions almost always share decisions. These common decisions may relate to anything from high level system design to actual choice of components. For example, almost all automobiles share decisions about the basic breakdown of functional units (engine, drive train, etc.); almost all intermediate cars share decisions about size and placement of engine, passenger seating, fuel tank, etc.; almost all General Motors intermediate cars share decisions about actual components (same types of fuel lines, piston rings, etc.).

By examining previous decisions, designers can decide whether they can re-use parts of an existing design, look ahead to see the effects that similar decisions have had in previous designs, and so on. To provide this capability, the system must present design information, including decisions and rationale, in an understandable way. Also, since the volume and complexity of the recorded information will rapidly grow beyond the limits of human comprehensibility, the system must provide mechanisms for locating and automatically recalling relevant designs.

For example, some of the decisions (including some of the abandoned alternatives) that led to the partial circuit board design shown in Figure 1 can be viewed in our Design Memory as shown in Figure 2[1]. The actual Design Memory itself would incorporate many designs, and thus be much larger and have a much more complex structure. The Design Memory structure will be a multi-dimensional space since decisions can be related by structure, function, behavior, etc. We believe that designers need direct access to the information in this Memory. But, because the Memory is so large and complex, they also need system-provided help in finding relevant places in Memory and in "navigating" the Memory once they find a relevant neighborhood. These ideas, and the facilities we are building to provide the required help for the designers, are discussed in Section 4.

In addition to these navigation tools, techniques for interactively "growing" the Design Memory are also being developed. The central tenet of these techniques is the generally passive nature of the acquisition. Memory growth will occur as a side-effect of using the system, augmented only when necessary by additional user interaction. A brief discussion of our Memory acquisition techniques appears in Section 5.

[1]This design protocol is due to Allen Sullivan, Arizona State University.

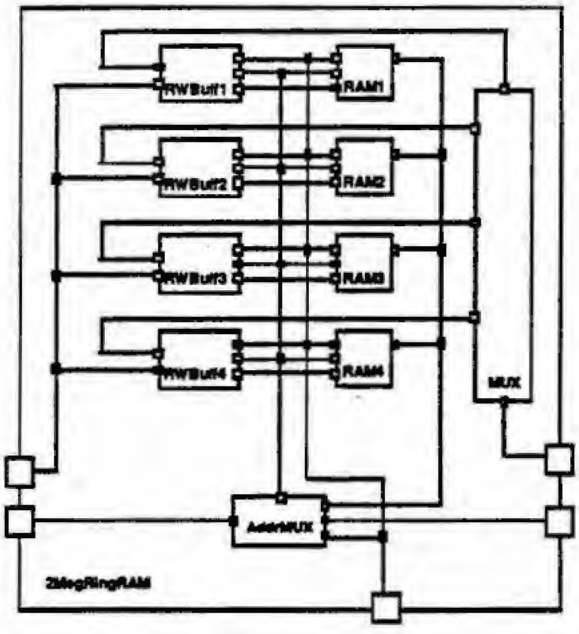


Figure 1: Part of a Circuit Board Design

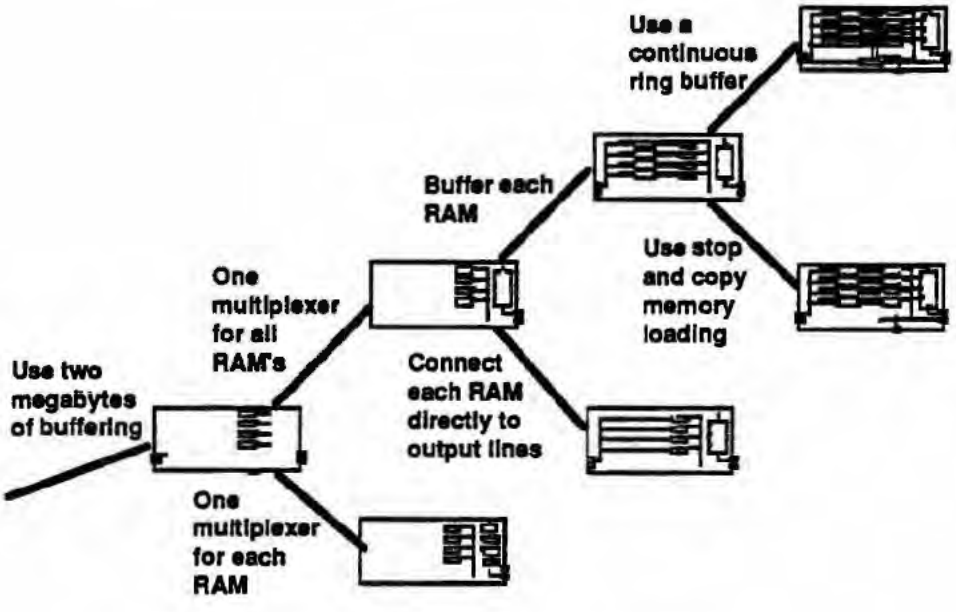


Figure 2: Part of a Design in Design Memory

3 Current Approaches

The only record of designs found in most environments – drawings and manuals – does not capture the reasoning that went into the design, and is not accessible through any but the most primitive recall mechanisms; we do not consider these to be design memories. However, several approaches are being applied to the creation of design memories.

3.1 On-Line Documentation

An obvious way to directly capture design knowledge beyond the level of drawings and specs is to have designers type in annotations explaining why they are making decisions, what alternatives they have considered, etc. These annotations can then be examined by later designers (or the same designers later on). The major advantage of this on-line documentation approach is that the designers' input can be free-form, governed only by their sense of what other designers will want to know about their designs (though requirements and restrictions can also be imposed externally as a matter of usage policy).

A major disadvantage is that it is very hard to get designers to do this work: it involves a great deal of effort (composing and typing text) with only indirect benefit – if that – to the designers who do the work. Combined with this is the sociological problem that people are reluctant to "go on record" with opinions that may be tentative or simply difficult to explain.

Furthermore, the advantage of "natural" input is also a major disadvantage. Since the rationale description is in terms of "uninterpreted" text and graphics, the system is unable to organize and find information, except in the most primitive ways. For large scale systems, this is a fundamental failing: the rationale documentation for a large system can be incomprehensible, even though each piece of information in the documentation is in itself perfectly comprehensible.

3.2 Electronic Notebooks

An approach to ameliorating some of these difficulties is the "electronic design notebook" [Lakin, *et al.* 89], [Bouchard, *et al.* 89]. These systems use sophisticated graphics and text manipulation capabilities to allow designers to use them as they would their notebooks, sketching and writing notes informally as ideas occur to them. Analysis software used by the designer is provided as an integral part of the system. The idea is to encourage desigers to record information in the system rather than on paper, thus capturing the information – at least in raw form – automatically. Designers are further encouraged to use the system by value-added features such as automatic cross-reference, drawing clean-up, and summarization, hypermedia-based interaction, and so on.

These systems introduce a significant advance over on-line documentation by drastically reducing the burden on users, essentially recording all information that is currently written down, and perhaps encouraging more information to be made explicit. The approach is still limited to recording what is customarily found in designers' notebooks, and by the sociological problems mentioned earlier if the information is to be used by anyone other than the original designer.

As with on-line documentation, a more serious limitation is introduced by the very "naturalness" that makes the approach feasible in the first place: it may be very difficult for designers to understand the contents of someone else's notebook, electronic or otherwise. In fact, there may be a loss here compared to more formal annotation schemes, because in the drive for flexibility, adherence to any externally understandable structure may be sacrificed. Of course, it is even harder for the system itself to understand the information, so little help can be expected in terms of automated retrieval and interpretation facilities.

This problem is recognized by developers of electronic notebooks, and the trade-off among total flexibility, user-provided structure cues, and system-provided structure is being explored. Nonetheless, the summarization and cross-reference capabilities offered by these systems are purely syntactic.

3.3 gIBIS

gIBIS [Conklin and Begeman 87] is an example of a tool that attempts to provide a structure for capturing the design process. That is, it delivers a structured view of design, in this case based on the IBIS (Issue Based Information Systems) model of design deliberation [Rittel and Kunz 70]. Designers structure their input in terms of the IBIS framework of positions, issues, and arguments. The purpose is to record decisions and alternatives – with their rationale – in an examinable structure. The proposed advantage is that the design process will be more efficient (continual rehashing of decisions is avoided, it will be easier to incorporate new members on to the design team, etc.).

From the point of view of design memory, the advantage here is the observable structure: decisions and their rationale are explictly linked to related decisions and rationale. The knowledge acquisition mechanisms – the methods for building the memory – retain the major disadvantages discussed earlier: the designer is still responsible for typing in all rationale information (and now also for structuring it in the IBIS framework) and the information is still uninterpreted text.

gIBIS does provide hypertext mechanisms for aiding the designers in navigating exisiting structures. While this is an improvement over the facilities of unstructured approaches, the developers of gIBIS point out [Conklin and Begeman, 1987] that a "context" problem still remains: designers still have considerable difficulty in understanding enough of a recorded design process to provide context for understanding the particular decision or decisions that they wish to examine.

3.4 Case-Based Reasoning

Case-based reasoning relies on the assumptions that certain clearly delineated parcels of knowledge can be determined *a priori* to be worth recalling later, and that these parcels can be recalled on the basis of certain of their features – i.e., that they need not be re-examined in detail in order to determine their relevance.

While these assumptions are often correct (we among many others have applied case-based reasoning in a variety of domains), we do not believe that they hold for design (*cf.* [Goel and Chandrasekeran 89]). For large design efforts, the whole design cannot be a case – it is too big to understand and may contain much that is irrelevant or incorrect under new circumstances. It is

also dangerous to define cases based on the structure of the designed artifact. For example, making cases out of components is often not useful because the choice of components is itself a decision that depends on previous decisions. That is, if the designer makes a different decision about functional breakdown, all of the previously defined component level cases might easily become irrelevant. Finally, basing cases on abstractions such as the "purpose" of a set of components suffers from the same problem: purpose is defined on the basis of other decisions about purpose; it is not an independent aspect of the design.

This tight interdependency of decisions also makes it difficult to pre-define a set of features that can he used to determine the relevance of part of the memory. It is very hard to predict in advance how a particular set of decisions should be accessed. Decisions based on the functional capability of components have equal "recall status" with decisions based on the weight of components, their reliability, and even extrinsic characteristics such as how many times the components have been used in delivered systems and their availability in inventory.

4 An Alternative Approach

We believe that designers must be able to access the Design Memory at the level of individual decisions. Rather than trying to retrieve cases, we aid designers by helping them to focus on a relevant "neighborhood" in the Design Memory network, and then helping them to navigate once they get there. We use the rich structure of interdependencies that made it difficult to carve the Memory into cases as the basis for hoth focusing and navigation.

4.1 Focusing

Each decision made by the designers places a constraint on the design. Some of the previous decisions recorded in the Design Memory will be compatible with this constraint; others will conflict. Thus, each design decision can he used to *rule out* (at least temporarily) the relevancy of part of the network. This process of elimination strategy can be combined with a network visualizer to produce an efficient interactive focusing process to implement "reminding".

As designers make decisions, starting with the initial specification or "design constraints"[2], they are shown the part of the Design Memory that is compatible with their current design. When the relevant portion of the Memory is large (e.g., during the initial stages of a design), the visualizer will present a high level view of the relevant network structure, showing its shape rather than its details. As further decisions rule out more of the structure, the visualizer will zoom in, eventually becoming a "browser" for the network at the individual decision level.

We believe that designers must play an active role in this reminding process. First, as a practical matter, the system cannot be expected to understand all relationships among decisions, nor to be complete in detecting and ruling out inconsistent parts of the network. Second, we believe that reminding should be a feedback process: if the designers find that a decision rules out a part of the

[2]Specifications and design constraints are just another form of decision, perhaps made by someone else (or by "nature").

network they are interested in exploring, they may wish to reconsider the decision; or they may wish to go ahead and explore a set of decisions they had not considered before. Much like human beings "trying to remember something", the system's focusing process is a proactive, constrained search through memory. The interdependency of the decisions constrains the search tightly, ensuring rapid focusing.

For example, designers would start their use of Design Memory by either entering or browsing for an existing set of initial design constraints (i.e., specifications) that approximate those of their current design problem. These would probably consist of system-level constraints like "only components qualified for spaceflight", "power dissipation less than 26 w", "weight less than 22 lb" – but the system could accept constraints at any level of detail.

For a large Design Memory, any small set of reasonable constraints is likely to be consistent with some subset of Memory. The size of this subset will clearly depend on both the constraints and the number of designs stored in the Memory. We will assume in this example that the first subset is too large to be comprehensibly navigated, as shown in Figure 3[3].

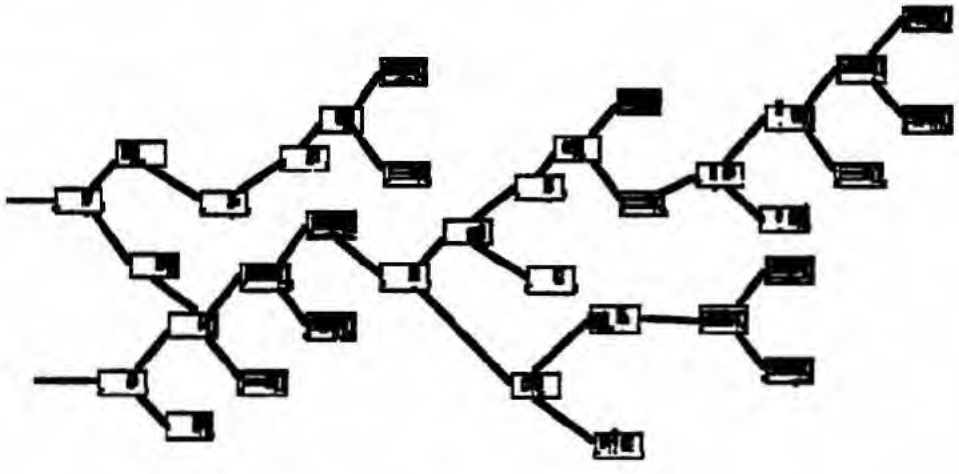


Figure 3: First Focus

The designers can enter more constraints or simply zoom into the structure – probably still leaving a fairly large subset, as shown in Figure 4. Here the decisions are at least visible, allowing the designers to see in detail the effect of their constraints on further focusing. For example, if the designers enters the constraint "high burst input data rates", the branch labelled *Add serial byte converter* is eliminated, since this decision and those following it are inconsistent with this requirement. An additional constraint "recoverability for any single point failure" would eliminate the *Use FIFO's* branch after *Buffer input*. Finally, the constraint "simultaneous I/O" would leave the subset shown previously in Figure 2.

[3]As mentioned earlier, these figures (Figures 2, 3, and 4) represent what the designers would see, not the actual Design Memory structure. The actual Memory (when "fully grown") will contain many overlapping designs: alternatives abandoned in one design may be successfully pursued in another; order-independent alternatives will be possible at many different places in the design; and so on. The Design Memory is therefore a complex multi-dimensional structure that must be filtered and interpreted to provide presentations that are understandable to the user.

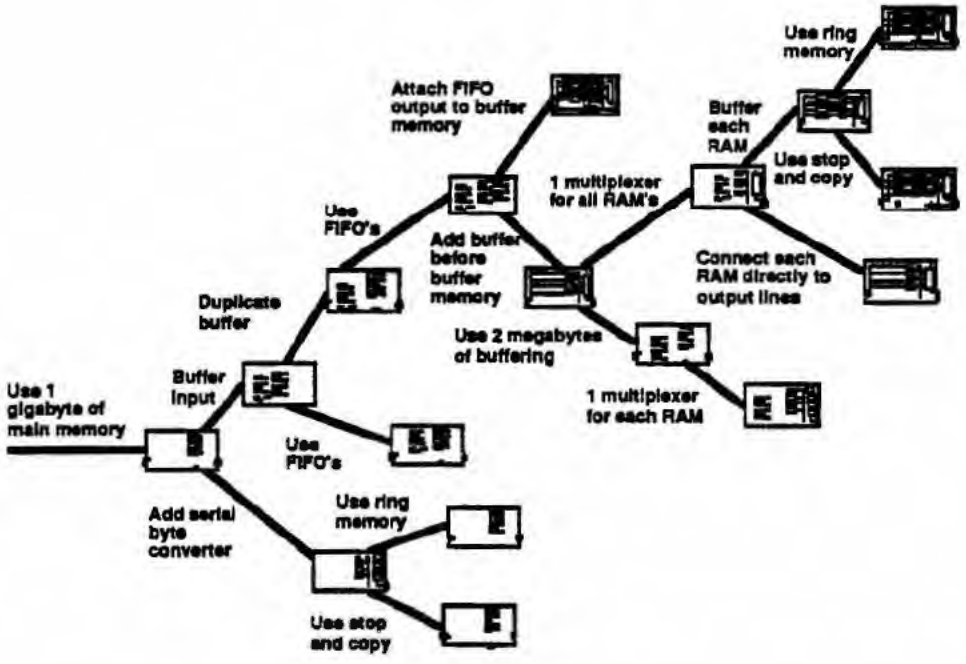


Figure 4: Second Focus

At this point the designer can begin the step-by-step examination of the Memory that we have been calling navigation.

4.2 Navigation

Once the visualizer has focused down to the level of individual decisions in the Design Memory, the effect of further designer decisions can be realized by navigating the explicit network structure. The designers may want to make one of the decisions they see in the structure. This corresponds to moving forward one link in the network. The system supports this activity through the network browser, providing facilities for selecting decisions and receiving explanations [in English] about them, and for moving through the network. In this way, the designers can thread through the Memory, exploring alternatives and choosing a design path.

The designers may of course want to make a decision different from any alternative they see in the neighborhood. This can come about for two reasons:

- the designers explicitly want to try a new alternative: their decision fits into the context established by the neighborhood, but the Memory has never seen it before;
- the designers want to make a decision that does not seem relevant to their current context.

The first case corresponds to growing a new link in the network. The system gives designers full flexibility in making design decisions that cause the system to create new links, except that the new decision must be consistent with the rest of the decisions made so far. Inconsistencies must

be resolved – either by changing the decision, or relaxing a previous constraint (which may cause re-focusing)

The second case implies that the designers need to stop navigating and re-focus on a different neighborhood in memory. This does not necessarily mean that the old focus is "wrong" and should be abandoned. The designers could simply have decided to work on a different aspect of the design. The old focus is thus remembered when the new one is established. The case of multiple foci is interesting: foci may converge later (implying that the designers have simply been working on different parts of a consistent design), or they may stay divergent (implying that there is no consistent way to reconcile the pieces). If the system can detect that a decision in the current focus is inconsistent with a remembered focus, it will warn the designers, who can then decide whether to retain the inconsistent pieces, drop one of the foci, or change one of their decisions. This provides an interesting paradigm for group design efforts.

5 Building the Memory

Starting with a built-in "seed", the Design Memory grows through interaction with the designers. As discussed in the previous section, the system will have a mechanism for incorporating new design decisions into the Memory structure. The mechanism is limited by the system's ability to see how to "link" the decision into the current network. In particular, the system must be able to determine to the greatest extent possible what new constraints the decision adds and whether those constraints are consistent with the constraints in the current focus.

We assume that the system will know the "type" of the new decision: there are certainly categories of decisions (e.g., "adding a component") that can be further specialized ("adding a buffer", "adding a 1MB buffer", and so on), with each specialization forming a type. Types that are built into seed memory will include information about how that type of decision affects and is effected by constraints (e.g., the amount of power and space consumed, input/output requirements, etc.). A new decision must specialize a known type at some level of detail, so the system will know at least something about how to link it into in the network. The more specific the type that is specialized, the more the system will know about the new decision.

Our goal is to make this memory building a *side-effect* of using the system; i.e., the designers' goal in using the system will not be to build the memory, it will be to use the memory for guidance and re-use as described earlier. It is the system that has the goal of building memory. We believe that both the users' and the system's goals can be met through the normal process of interaction with the designers: the system presents the designers with relevant previous design decisions, and the designers either incorporate these decisions into their designs or explicitly make new decisions in their stead. If a designer makes a new decision, the system grows its memory by relating this decision to the design decisions it already knows about.

6 Status

We have implemented a preliminary demonstration of Design Memory, including primitive visualization, explanation, and constraint checking capabilities. This effort is part of a larger Knowledge Centered Design Project at Lockheed [Kellogg, *et al.*, 89], [Foyster 84]. We are currently experimenting with real designs from the domain of complex electronic circuit boards (the component shown in Figure 1 is a small part of such a board). Besides work on implementing the facilities described in this paper, we are currently creating the seed Memory of "understood" designs (i.e., designs represented in terms of decisions) that will both test and show the applicability of our approach.

This implementation uses the LOOM knowledge representation language [MacGregor 89] to maintain a representation of the Memory consistent with a designer's specifications. To examine a specific design in the Memory we are using NVisage [Weber 90], also being developed at the Lockheed AI Center. The NVisage system concurrently manages and displays representations for distinct aspects of designs such as physical structure, logical structure, simulated values, and testability.

7 Discussion

We believe that our methods are suited to any domain in which reminding occurs in the context of a highly interdependent structure. For the techniques to work, the interdependencies must be understood and representable. Complete coverage of interdependencies is not required (indeed completeness is almost certainly not feasible in any real domain), but there must be enough coverage to make focusing possible.

In Section 5 we discussed the system's ability to incorporate new decisions into memory. We believe that it will be possible to greatly expand this ability through a form of analogical reasoning. Even though a decision is "new" in one context, it may be similar to a decision in a different context. We envision a four step process for using this knowledge to learn more about the new decision: (1) finding the other situations in Memory in which a decision of the same type has been made; (2) comparing the constraints of the current situation with the constraints of the other situations to select the situations that are "most compatible" (i.e., that can be made consistent with the current situation using the least number of best understood constraint relaxations); (3) creating a mapping between the current situation and one or more compatible situations; (4) presenting the system's assumptions about the new decision to the designers by explaining the analogy and the constraint relaxations on which it is based.

8 References

Kevin Ashley and Edwina Rissland, "A Case-Based Approach to Modelling Legal Expertise", *IEEE Expert*, Vol. 3, No. 3, 1988, pp. 70-77.

Eugene Bouchard, *et al.*, "The Engineer's Associate", *Proceedings of the Fourth Annual Lockheed AI Symposium*, Calabasas, CA, March, 1989.

Jeff Conklin and Michael Begeman, "gIBIS: A Hypertext Tool for Team Design Deliberation", *Proceedings of Hypertext '87*, November, 1987.

Francois Daube and Barbara Hayes-Roth, "A Case-Based Mechanical Redesign System", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Palo Alto, 1989, pp. 1402 - 1407.

G. Foyster, "HELIOS User's Manual", Stanford Heuristic Programming Project Report No. HPP 84-34, Stanford University, 1984.

Ashok Goel and B. Chandrasekaran, "Use of Device Models in Adaptation of Design Cases", *Proceedings of the Case-Based Reasoning Workshop*, Morgan Kaufmann, San Mateo, May, 1989, pp. 100 - 109.

Charles Kellogg, William Mark, and Joseph Sullivan, "Interactive Acquisition and Explanation of Design Knowledge", AIAA 1989 Aerospace Sciences Meeting, Reno, Nevada, Jan., 1989.

Fred Lakin, *et al.*, "The Electronic Design Notebook", *VISUAL COMPUTER*, Springer-Verlag, 1989.

Robert MacGregor, "LOOM User's Manual—Draft", USC/ISI Technical Report, November, 1989.

Jack Mostow and Mike Barley, "Automated Reuse of Design Plans", International Conference on Engineering Design, Boston, MA., Aug., 1987.

H. Rittel and W. Kunz, "Issues as Elements of Infromation Systems" *Institut fur Grundlagen der Planung I.A. Working Paper #131*, University of Stuttgart, 1970.

Katia Sycara, *Resolving Adversarial Conflicts: An Approach Integrating Case-Based and Analytical Methods*, PhD Thesis, School of Information and Computer Science, Georgia Institute of Technology, 1987.

Jay Weber, Brian Livezey, and James McGuire, "Simulation by Cooperating Distributed Representations", Proceedings of the 1990 Workshop on AI and Simulation, Boston, MA., Aug., 1990.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

The Ethnological Method of Participant Observation as a Technique for Manual Knowledge Acquisition

MARY A. MEYER*, RICK R. PICARD, 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*

We describe a recent study to illustrate how the anthropological method of participant observation can be used in manual knowledge acquisition to develop models. Participant observation occurs when the researcher not only observes but participates in the activities of those persons being studied. By participating, the knowledge engineer is able to use his or her responses to the situations as data to formulate hypotheses about the experts' behavior and its meaning. We illustrate how these hypotheses can be checked through other knowledge acquisition techniques, such as thinking-aloud protocols and structured interviewing, and through the use of statistical experimental design.

1. Introduction

In traditional participant observation, the researcher is not a detached observer but a participant in the activities of those being studied. Participant observation is generally defined as a

> field method whereby the ethnographer is immersed in the day-to-day activities of the community being studied .... The objective of this method is to minimize the presence of the field worker as a factor affecting the responses of the people and to provide a record of observed behavior under varying conditions (Hunter and Whitten, 1976, p. 294).

Participant observation has traditionally been seen as means of improving rapport with the informants or, more generally, those being studied (Kottak, 1974, p. 496). In addition, ethnologists (cultural anthropologists) have found that participant observation has frequently stimulated them to generate hypotheses which could then be examined through the use of more structured methods as interviewing and tests. For example, after participant observation the researcher might hypothesize that members of one group exhibited one behavior and those of another, a slightly different one in the same situation.

According to some anthropologists (Crane & Agrosino, 1974, p. 63), participant observation is not so much a technique (i.e., a program for action) as a *state of mind.*[1] By this they mean that one does not *do* participant observation in the sense that one does a kinship chart or opinion survey. Instead, participant observation is more a framework for living in the field. Direct experience of other's way of life is viewed by many ethnologists as the most effective way of reducing one's natural ethnocentric biases. A researcher's ethnocentrism arises from being raised in a society that possesses particular ethical, social, work, religious, and sanitary values.

[1] While we agree with this characterization of participant observation, we refer to it as a *technique*, mainly to distinguish it from the automated *tools* of knowledge acquisition.

Ethnocentrism results from the researcher seeing things in terms of the culture of which he or she is a product and therefore tends to view as superior, or at least acceptable.

Our experience leads us to propose that participant observation can be applied to manual knowledge acquisition to meet these purposes or to provide the following five benefits.

First, the observational aspect of this manual technique puts the knowledge engineer in place for receiving the meaning that the experts' attribute to their own actions. Many psychologists, psychiatrists, and practitioners of knowledge acquisition believe that the drawing of distinctions is at the base of what humans experience and learn. Observation places the researcher in in a convenient position to hear the insiders' descriptions of distinctions that they are making. Thus, participant observation allows the researcher to learn what meaning the insiders' actions hold for them in the insiders' own words. (Note that the insiders could include domain experts in interaction with others or the intended users of the knowledge-based system.)

Second, observation allows the knowledge engineer to view *knowledge rooted in activities*[2] , and, thus, a chance to verify the experts' descriptions of their problem-solving processes or to gather information that the experts could not articulate. This contextual view of knowledge portrays knowledge as actions taking place in the "stream of experience which envelops the person"; a stream of experience which is defined to include "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." (Kwasnik, 1990, p. 124) In other words, this view portrays "peoples' knowledge as not being just in their heads but in the environment in which they are embedded."[3]

Frequently the experts have no awareness of or ability to describe some aspect of their work, because they have been asked to so out of context. Other times, experts provide a description of their actions which do not match their actual performance, particularly in the sequence of actions (Breuka & Wielinga, 1987, p. 36). Observational techniques are therefore often necessary to obtain information on expert knowledge.

In addition, observing experts in situ offers the knowledge engineer unique insights into the social process of expertise (cf. Gaines, 1989), that is, how expertise evolves through the individual expert's receiving of feedback from other experts and clients. Understanding how experts update their knowledge is of practical use, if we are to design knowledge-based systems with similar capabilities, as proposed by Gaines (1989). Additionally, participant observation allows knowledge engineers to examine how experts reach consensus or settle disagreements. These observations may provide future guidance or algorithms for combining and integrating differing expert knowledge in knowledge-based systems.

Third, the participation component of participant observation allows a deepening of the knowledge engineer's understanding of the insider's activities and a greater rapport. For example, participation would enable the knowledge engineers to put themselves in the user's shoes to learn their needs and preferences for the system. As mentioned earlier, many ethnologists believe that the most effective means for minimizing one's own ethnocentric bias is participation in the insider's activities. Knowledge engineers, as human beings, tend to interpret new information in light of what they already have learned and have come to value. Thus, knowledge engineers are prone to misinterpreting and misrepresenting the expert's verbalizations or behaviors. Participant observation provides a check on the knowledge engineer's misunderstanding of the experts' reports of their actions.

---

[2] This term is used by Button, Sharock & Anderson (1990, p. 13).

[3] This is quoted from Gaine's summary talk at the European Knowledge Acquisition Workshop 1990, Amsterdam.

Fourth, participation aids knowledge engineers in constructing personal models of the objects and events within the domain which, in turn, can be used in building the knowledge base. Clancy (1989, p. 10) has likened knowledge bases to qualitative models. He further defines qualitative models as describing "systems in the world in terms of causal, compositional, or subtypical relationships among objects and events." Following this view, then, knowledge engineers are likely to need a variety of techniques and tools to assist them in building models. We propose that participant observation provides the knowledge engineer with ideas about relationships between the elements of the domain through an unusual avenue—*participation*. Participation allows the knowledge engineer to receive his or her own responses to situations as data on relationships.

For example, suppose that the knowledge engineer had been internally evaluating the credibility of threats of violence while the expert assessors went through this same process in group discussions. The knowledge engineer may have observed after participating in the team assessment of a threat judged credible that she/he was subsequently more willing to consider other threats as serious. The hypothesis from his/her own reaction could have been that assessors who had been involved in evaluating one or more threats judged credible would be more likely to consider later threats as credible

Hypotheses about relationships can be tested through the use of additional methods before inclusion in the knowledge base.

Fifth, the performance of participant observation, in general, serves to familiarize the knowledge engineer with the domain and experts. For example, Boose and Shaw (1989, p. 6-62) mention participant observation as something that the knowledge engineers can do before their first interview with a source expert.

How participant observation is enacted may differ depending on the purposes for which it is being used. For example, if participant observation is simply being used to familiarize the knowledge engineers with the domain, it will has a different flavor than if its purpose is stimulate their formulation of hypotheses for model development.

This paper is organized as follows. In section 2, we present background information on participant observation: the history of its practice in cultural anthropology, the type of thinking that it involves, and a comparison of it to other related knowledge acquisition techniques, such as on-site observation or apprenticeship learning. In section 3, the preliminary study of threat assessors is described. Through this study, we illustrate how to use participant observation with two knowledge acquisition techniques—concurrent verbal report and structured interviewing—and with statistical experimental design. In the threat assessor study, participant observation was used to provide the knowledge engineer with an insider's insights and to stimulate the formation of hypotheses. In section 4, we describe some of the assumptions about knowledge that are implicit in participant observation and offer guidance on the use of participant observation in knowledge acquisition. Section 5 is a summary.

2. Background

2.1. HISTORY OF PARTICIPANT OBSERVATION IN ANTHROPOLOGY

The development of participant observation is formally credited (Hunter & Whitten, 1976, p. 294) to Brownislaw Malinowski, a Polish anthropologist, who began his field studies in the early 1900's. He and other pioneering anthropologists initiated modern field work, the practice of residing in the community of study for extended periods of time. These anthropologists went beyond relying on second-hand, and usually erroneous reports of the people from traders and

missionaries by actually living with the natives and observing them. However, this trend can be viewed as a formalization of an earlier practice.

Before the field of anthropology existed, Crane and Agrosino (1974, p. 64) propose that participant observation was being practiced on the American frontier. They cite the example of Henry Schoolcraft, who arrived in the frontier region of of the upper Great Lakes in 1822, to work as an Indian agent among the Chippewa (Ojibwa). Crane and Agrosino trace Schoolcraft's coming to understand the Indians by marrying into one of their families and living with them.

> He began to understand that the errors and distortions about Indian cultures (from the "bloodthirsty savage" stereotypes of the frontiersmen to the "noble savage" romanticism of the European philosophers) were a result of constant trading in second-hand information. By living with the Indians, not merely look at them from the outside, Schoolcraft learned ... that the Indians created myths, tales and legends that they told for both amusement and instruction. Although today we take it for granted that even the most technologically primitive people can possess an artistic tradition (be it graphic or verbal) of great richness and complexity, this was a revelation of the first order in Schoolcraft's day. And, more important, it was a revelation that could never have come to him in his agency office—discovering the "secrets" of the Indians required him to live among them as a friend who took part in their daily lives, participating in ordinary food-gathering forays as well as in elaborate rituals, observing them from the vantage point of one who played as active a part as possible in their "real" lives (Crane & Agrosino, 1976, p. 64-65).

2.2. PARTICIPANT OBSERVATION AS A WAY OF THINKING

Crane and Agrosino have further described participant observation as a state of mind. We add that we consider participant observation to be a way of thinking, distinct from that used in the research associated with the physical sciences. Belenky, Clinchy, Goldberger, and Tarule (1986) who have done an epistemological study of women's ways of thinking about the world, describe participant observation as a sophisticated form of *connected knowing*.

Connected and separate knowing are given as two general types of thinking. Separate knowing is defined as ways of thinking that have been associated traditionally with scientific research, namely empiricism. In separate knowing, the individuals believe in absolute truth or procedures for obtaining objective facts. Separate knowers strive for objectivity by attempting to detach themselves from what is observed so that they might avoid projecting their own thoughts into the observed. In this way, separate knowing contrasts with connected knowing; connected knowers seek to enter the observed's world to understand other people's ideas in those people's terms rather than their own. The experience of connected knowing is viewed as a union between the learner and what is to be known. It is perhaps for this reason that connected knowers frequently describe their experience as *immersion*.[4] In sum, connected knowing is based on empathy, trying to approximate other's experiences as a means of understanding others' views in their own terms. Belenky et al. emphasizes this quality of participant observation by describing it as "technique of disciplined subjectivity" requiring that the researchers "systematically empathize with the participants" (1976, p. 226).[5]

---

[4] For instance, in participant observation, both Crane and Agrosino (1976, p. 63) and Hunter and Whitten (1976, p. 295, p. 295) describe the ethnographer as being *immersed* in the day-to-day activities of the people being studied. Another illustration of the quality of immersion is given by a famous painter (Quiller, 1989, p. 226): "Once the subject has chosen me, I try to get as involved as possible with it. I let my emotions flow toward the subject and get completely *immersed* in it. As I paint, I become part of the subject, rather than a spectator looking in."

[5] We note that the connected way of knowing is not limited to participant observation but exists in some forms of pyschotherapy in the practice of imitating (mirroring) the individual's expression, breathing, and posture as a means of gaining insight into the client's feelings.

Belenky further describes participant observation as involving observing oneself, the observer, in the situation; as bringing personal issues to consciousness so as to be aware of their influence on one's observations. In sum, in participant observation, "the researcher, uses him or herself as a tool for the investigation. The researchers use their reactions to understand other's views and to formulate hypotheses about the other participant's reactions." (Belenky et al, 1976, p. 226).

2.3. RELATION OF PARTICIPANT OBSERVATION TO OTHER KA PRACTICES AND TECHNIQUES

While participant observation has been proposed as knowledge acquisition technique (Boose, 1989, p. 12), it has not frequently been used, perhaps because it is still relatively unknown in the knowledge acquisition community. However, participant observation may have been employed de facto in several cases where the knowledge engineers were members of the expert population whose knowledge was being modeled (e.g., with the physicians who worked on developing EMYCIN). This trend for the researcher to be a native has also emerged in ethnology, such as with anthropologists who are Tewa Indians doing ethnographies of the pueblos in which they were raised (e.g., Ortiz, 1969).

Participant observation most closely resembles two techniques in knowledge acquisition—on-site observation and apprenticeship learning/teachback interviews. On-site observation is described as watching the expert solve real problems on the job (Waterman, 1986, p. 158). Thus, on-site observation likely differs from participant observation in that passive observations do not involve participation nor *systematic empathizing*. Apprenticeship learning or teachback interviews (Boose, 1989, p. 12) involve learning from one expert at a time and having the knowledge engineer demonstrate his or her understanding by solving a problem. In contrast, participant observation commonly involves learning from multiple experts' interactions, some of which may not be directed toward problem solving so much as building expertise (e.g. through arguments among experts). In this paper, we consider apprenticeship learning or teachback interviews to be a subvariety of participant observation. Indeed, participant observation has been defined as "when the knowledge engineer becomes an apprentice or otherwise participates in the expert's problem solving." (Boose, 1989, p. 12)

In ethnology, participant observation has rarely been used by itself but is combined with more analytical and manipulative methods. For this reason, in knowledge acquisition, we would recommend its use in combination with other manual methods of knowledge acquisition and statistics. Care is needed in combining participant observation since, as Shaw and Woodward (1989) have noted, ad hoc combinations of knowledge elicitation techniques have led to a situation where these may contain conflicting assumptions as to the nature and characteristics of knowledge. (See section 4 for further discussion of the assumptions inherent in participant observation.)

3. Sample Application of Method

3.1. THE SAMPLE DOMAIN AND POPULATION

The domain involved experts' assessments of threat messages. (Some details on this domain are omitted due to the sensitive nature of the service that these experts perform.) A *threat message* is a communication that threatens or warns of someone's intent to commit a particular act. Often the message follows an extortion format of *give me this or else*. Things that individuals have tried to extort have been large sums of money, political actions, environmental actions, and media coverage. Sometimes the demand is not stated but only implied; other times, the extortion part of the threat is absent. For example, if the individual states that he or she will commit some act

regardless, the demand is missing. The extortion may also be missing in situations where the writer is warning of other's potential acts. The threatened act has ranged from bombings, contamination of a water supply, to the destruction of a power plant. Threat messages can and have been, delivered by different media. The message may be written (e.g., a letter), visual (e.g., a diagram or video tape), or aural (e.g., a tape recording or a telephone call). The written transcripts of the threat messages have ranged in length from being one line to many pages.

Assessors are persons who have been invited to serve on small assessment teams because they possess some specialized knowledge relating to the threat message or to the perceived personality of the message creator. These assessors are a dedicated and close-knit group. The assessors have explained their rapport as stemming from their feeling of having undergone unique experiences in the threat assessment process. Small teams of assessors are headed by a *team leader*, who has many years of experience in doing these assessments.

The process of threat assessment begins when the assessment coordinator receives a threat message from an organization in the United States government. The assessment coordinator notifies the assessors and generally oversees the assessment process (e.g., when and where the assessors are to convene).

When the assessors receive a written transcript of a threat message, they have a period of time for reading the message to themselves. This time can varies from a few minutes to many hours depending on when the assessor receives her or his copy of the message and when the team must convene. The individual reading of the message frequently occurs in the place where the rest of the team will assemble, especially when there has not been much lead time as to when an assessment is needed. In the latter case, the team members enter and quickly read the message before the team discussion begins. During this reading, the assessors try to focus on the message and use their own means of evaluating it. From the reading, the assessors may form an impression of whether they, as individuals, consider the message to be credible or not.

When the team is assembled, the assessors bring their thoughts and questions on the threat message to attention of the group. While the team is gathered, the assessment coordinator may provide additional information on such topics as the deadline for the assessment, and whether more information is expected to arrive shortly. The team leader may also ask the assessors to address specific questions relating to the threat. The team discusses the individual's ideas in an informal manner until all the questions have been raised or the deadline for conclusions has arrived.

The team leader asks for the team's conclusion, given in terms of three ratings of credibility. There may be more discussion at this point, if the assessors do not agree on a conclusion. After consensus has been reached, the team leader will request a rating (one of three categories) on how confident the team is of its conclusion. Often the team leader will read aloud his or her notes on the team discussion. After team members have approved the team leader's account, the team's conclusion, confidence rating, and rationale is sent to the office requesting the threat assessment.

The events following an assessment have indicated that the assessors generally make accurate predictions. The vast majority of threats rated not credible have not been followed by the threatened events. The few threats which have been given a credible rating have usually been followed by actions similar to those threatened.

3.2. PURPOSE OF THE PROJECT

The goal of this preliminary study was to make explicit the assessors' unconscious patterns of thinking and interacting. The researchers interpreted this goal to include the tasks of: (1) describing how the assessors reached an assessment conclusion/confidence rating, and (2)

determining which factors in the threats messages, the assessors themselves, and the team meetings affected this conclusion/rating.

The goal of making these patterns explicit and documenting them was to serve two purposes. First, this information, in the form of a primer, would be used to train new assessors so that they would be more prepared and confident in doing their first assessment. Second, this information would allow assessors and others to understand and improve the assessment process. Previously, new assessors had *learned by doing* because there was no formal description of how the assessors evaluated threats. Such a description was lacking because the assessors could not easily describe how they evaluated threats. In particular, they were unable to describe the interactions that occurred in the team discussions. It was possible that the documentation of the assessors' problem solving might serve a third purpose in the indefinite future—that of providing the foundation for the development of a knowledge-based or expert system.

3.3. WHY PARTICIPANT OBSERVATION WAS NEEDED

Participant observation was needed, in addition to some of the more commonly used knowledge acquisition techniques, because the assessors reached their assessment conclusions through an interactive group process. The assessors could not describe their group decision making process in any detail. Thus, some observational technique was needed. (Videotaping was not a viable option because we would have needed advance notice of the meetings to arrange use of the equipment.) While any observational technique would have been helpful, we expected the participant aspect to allow us to better understand the process and the assessor's view of it.

3.4. AN OVERVIEW OF THE TECHNIQUES

3.4.2. Preliminary unstructured, focused, and structured interviews

The researcher interviewed the assessment coordinator, a team leader, and several assessors using these three techniques. In the unstructured interviews, the interviewer spontaneously asked questions and recorded the answers; in the focused interviews, the interviewer directed questions toward a number of topics; and in the structured interview, the interviewer elicited answers to a formal list of questions (cf. Boose, 1989, p. 12 on types of interviewing).

This period would have been a good time for doing participant observation as a means of familiarizing the researcher with the experts and the domain. However, the assessors had not been requested to give a threat assessment at this time and thus were not assembled from their regular places of work to be observed.

From the combination of these three interviewing techniques, information was gathered on the domain which was used to orient the researcher and to allow her to plan the later knowledge acquisition methods. As a result, the researcher decided to use three techniques for the next stage of the study: the *thinking-aloud* protocol (concurrent verbal reports), structured interviewing, and participant observation. The plan was to visit the assessors' places of work and have each of them *think aloud* as they, individually, read through old threat messages. (Transcripts of old threat messages were used for the verbal report technique, as opposed to invented new ones, because it would have been difficult to realistically simulate new threats and to anticipate the effects that their artificialities could introduce.) Then, after the assessor finished with his or her verbal reports, the researcher planned to ask a set of interview questions to learn about differences in the assessors and in the threat messages. The researcher planned, in addition, to observe any convening of the assessment teams, if these occurred during the period of study. The records of past assessments indicated that a least one team assessment was likely to occur within this period of time.

3.4.3. Pilot concurrent verbal reports and structured interviewing

Verbal reports, often called protocol analysis, is a technique for obtaining subjects' verbal descriptions of their thoughts. Verbal reports have been refined and popularized by two psychologists, Ericsson and Simon (1980; 1984). This technique is usually administered to one person at a time. The person is presented with a problem in written form and instructed to *think aloud* as he or she works at assessing it. Not all experts are able to verbalize their thoughts nor are all technical problems amenable to direct verbalizations. The knowledge engineer generally does not question the expert but records all the expert's words. The verbal protocol is based on the belief that subjects can access the thoughts that are currently in their short-term memory, convert these into verbal form, and vocalize them to the researcher. Ericsson and Simon claim that "cognitive processes are not modified by these verbal reports." (1984, p. 16).[6]

There are two types of verbal reports, *concurrent* and *retrospective*. With the concurrent verbal report, the experts verbalize their thoughts *as* they are having these thoughts; with the retrospective type, the experts describe them afterwards (e.g., immediately after solving the problem or some time later.) We chose the concurrent type of verbal report because it is more likely to elicit the expert's cognitive processes while these are still in the short-term memory. Thus, the experts' descriptions of these processes are expected to be relatively complete.

The verbal report technique was pilot tested, administered to a sample of the assessor population, for several reasons. One reason was to determine if the larger population experts would likely be able to *think aloud*. Another reason for pilot testing the verbal report was to gather information on what some assessors considered in reading the threat messages, particularly those characteristics that seemed to affect their verbalizations about whether the threat seemed credible or not. Because we initially knew little about the domain, we were eager to learn how the assessors categorized and thought about different threat messages. The pilot verbal report and structured interviewing were to provide us with a list of distinctions that the experts made.

These distinctions would then be used to set up an experimental design for the later performance of these same two techniques (i.e., we would select assessors and threats according to their possession of these traits and assign assessors to specific threats to test the effects of these traits.) For example, the assessor's words, *This threat is no longer credible*, given below imply that the assessor was implicitly assuming that the threat was credible until a piece of information toggled his evaluation over to *not credible*. Other assessors seemed to make the opposite assumption during their verbal reports. This trait was labeled *predisposed to consider a threat not credible* and was used to assign pairs of assessors (one who could be categorized as *yes* and one as *no* on this trait) to each threat.

The pilot sample of assessors (over one third of the total population) were presented with a packet of written materials. The packet include the transcript of a recent threat message and its cover letter. The cover letter described to whom the threat message was addressed, how and when it was sent, and when it was received. Our objective in the verbal report was to have the assessors give first priority to working through the problem and to simply voice their thoughts as flow of consciousness. The interviewer told each pilot assessor:

---

[6] The cognitive processes that the individual undergoes in reading text have been described by Bower and Morrow (1990). First, readers translate the surface form of the text into concepts. They then use their knowledge of the world to identify traits of the characters and the settings, to link expressions belonging to these, and to draw inferences about the causal relations among the action sequences of the narrative. Thus, the reader forms a mental representation of the situation and actions being described. The reader then uses this mental model to interpret and evaluate later statements.

... pretend that you have just received this threat and have your usual amount of time to think about it before your team starts discussing it. To this end, I have changed the old dates on the threat. As you work through this threat, try to think outloud.

The assessor would frequently then flip through the pages of the threat assessment and look more closely at the information on the first page, particularly on when the message was sent, who sent it and to whom, and the deadline on the assessment. At this point, the assessor might mention a first impression of the threat as part of his or her thinking aloud. The assessor then might underline a section of the threat message and state the thought that it triggered. After repeating this procedure several times, an assessor would sometimes would point to a phrase and say *this does it. The threat is no longer credible. Unless I hear something that makes me change my mind in the team meeting, I'm going to say that this is a hoax.* Frequently, after the assessors finished skimming the threat message once, they would go back and read it for anything that they might have missed. Often on completing the threat, the assessors would volunteer their impression of the threat's credibility. Frequently the assessors would frame their impression in terms of the conclusions and confidence ratings used during the team meetings. These verbal reports lasted from 30 to 90 minutes.

Pilot Structured interviews were conducted immediately after the expert finished with the verbal report. The list of questions was pilot tested on over one third of the assessor population for two reasons: First, as a check on the clarity or the questions (e.g., did the experts interpret these questions in the way that the creator of the questions meant) and second, to learn of distinctions that could be used to form the experimental design for the later structured interviews. For example, the pilot sample was asked what made some threats seem more credible to them than others. Responses to this question provided us with factors for selecting threats for the experimental design that would represent the variety of threats that assessors were likely to see. We thought that different thought processes might be stimulated by some of these traits and wanted to gather data on the full range of these experts' problem-solving processes.

The assessors were interviewed using a list of questions. For example, if they had not volunteered their rating of the threat's credibility and of their confidence in their conclusion, they were asked to supply these. These data became major variables for the analysis. Some of the other questions asked were: whether they had assessed this threat earlier when it actually occurred (although the experimental design tried to limit this), which teams they had served on, and whether they had any questions on the interview process. This final question was considered especially important in the pilot interviews as a check on the clarity of the interview instrument. That is, if many assessors commented unfavorably on some aspect of the interview during the first interviews, it would have been redone for the experimentally designed version. These pilot interviews generally lasted about 30 minutes.

In addition, all of the assessors, whether they were part of the above-mentioned pilot sample or not, were asked an additional set of questions to allow us to assign assessors in the experimentally designed structured interviews which were to follow. Among the questions asked were: whether they were predisposed to consider a threat not credible until shown otherwise, and whether they had ever assessed a threat which their teams had given a credible rating. The latter was asked because the researcher hypothesized that those assessors who had assessed threats labeled credible might be more predisposed to consider all threats credible, at least initially.

3.4.4. Participant observation of the team meetings

For the researcher to have the opportunity to do participant observation, she had to be notified when a request for a threat assessment was received. The researcher asked the coordinator to call

her as he would any assessor when a request arrived. (In this study, the experts' daily activities did not generally include threat assessments; assessors typically only did threat assessments several times a year. Thus, in this study, participant observation could not be done daily. Therefore, the assessors were not so accustomed to the researcher's presence during participant observation that the effects of her presence could be dismissed.)

Three team assessments occurred during the period of study. Each time, the researcher accompanied other members of team into the meeting room. The assessment coordinator or team leader made an announcement that the researcher would be participating in the assessment. The researcher generally sat with the assessors, as opposed to behind them, and tried to go through the mental processes of assessment. However, the researcher did not verbally give an assessment conclusion because of her lack of expertise and the ramifications of these conclusions.

Three convenings of the teams were observed for threats that ranged from being a crude four-line bomb threat with no demands to a literate letter warning of an attack on a power plant. These team meetings lasted between two and four hours.

As mentioned earlier, participant observation gives a researcher an insider's view. For example, the researcher learned that the team members thought that they began their meetings with what one member termed *wild thinking;* with considering *the possibility that the threat message may be credible but written by someone who was stupid, ignorant, or crazy* (and thus, seem incredible when read from a logical point of view). After this stage, the team members saw themselves as examining the message from a more critical and structured viewpoint to provide their rationale for their subsequent conclusion and confidence rating.

As mentioned earlier, a major benefit of participant observation is that it stimulates the researcher to formulate hypotheses and personal models. In this case, the knowledge engineer needed ideas on how the individual assessors arrived at an assessment conclusion during the team meetings. By participating, the researcher formed her own model of how the assessors updated their individual views of the threat message during the discussion by team members. Like the individual team members, the researcher had previously read the threat message and formed her own impression as to whether it was credible or not and for what reasons. During the team discussions, the researcher found herself listening to a team member's view about the credibility of the threat and internally evaluating both what was said and the source. For example, researcher first judged whether the team member's view was coherent with her's and then whether the team member had expertise in the area in question. If the team member's comments made sense given the researcher's view of the threat and if the team member was considered qualified in the area (e.g., bomb design), the researcher was more likely to adjust her previous assessment than just disregard the information. In the researcher's case, a kinesthethic representation was used internally to do this mental adjustment. That is, the threat message was *felt* to move from the area of credible to the incredible or vice versa.

Support for this researcher's model of this mental process was found, in part, in subsequent verbal reports and structured interviews. Some assessors described feeling a scale (a balance) which tipped to the credible or incredible side with the receipt of new information. Others explained that they pictured debit and credit columns with each relevant bit of information being recorded in the black, designating credible, or red, incredible.

In this study, participant observation could not be used to verify the experts' descriptions of how they reached a consensus conclusion because they could not supply such information. However, participant observation *was* used to obtain information on a means by which the assessors were *not* reaching consensus. Consensus is often fostered by a group think situation where one or a few dominant individuals lead the thinking of the others. Individuals have been

found to acquiesce to other's beliefs without thinking to the detriment of the quality of the final decision. This phenomena has been termed the *follow-the-leader*, or *bandwagon effect*. Because of the potential for the group think effect, the researcher looked for signs that the assessors' thinking was being led consciously or unconsciously by one of their number. Signs of this problem are the lack of any dissenting opinions and body language indicating that the members are deferring to someone (Meyer, Booker & Bradshaw, 1990, p. 21). These signs were not observed, nor did the researcher see any other indications of one assessor influencing the others.

In addition, participant observation aided in learning some of the practical means by which the team members reached a consensus conclusion. In several instances, the individual's mental model of the threat (the actors, their intent, the setting, etc.) changed as a result of the team's discussion. However, sometimes an individual did not agree with the other's views. In one instance when an individual disagreed with the shared view at the time when the conclusion needed to be made, the team members agreed upon a conclusion by compromising on their confidence rating. That is, the conclusion of the majority on the threats credibility was recorded but the confidence rating was adjusted downward to reflect the one member's differing view (and the possibility of that member being correct and the rest being wrong).

3.4.5. Experimental design

Most generally, statistical *experimental design* is defined as the "planning of experiments. Such experiments are studies in which the experimenter deliberately controls certain factors ... that may influence the outcome of the experiment." (Tietjen, 1986; p. 67). The experimenter then observes or measures the results.

We used the experimental design in this study because we wished to test whether particular factors were linked to assessors' ways of thinking or to their assessment conclusions and confidence ratings. Our project differed from those studies of human behavior to which experimental design is more typically applied (e.g., clinical trial work that involves responses to medical treatment or market research work regarding people's activities/opinions). First, the number of relevant experts in this project was small, and, second, there were a comparatively large number of factors whose effects we wished to examine. These aspects of the project posed problems to the experimental design and the design required extra effort; however, we decided to use this technique because of the desire to formally test for statistical relationships. While in this preliminary stage of the research, the experimental design was used to test for factors found in the verbal reports and structured interviews, we hope to apply this technique later to test hypotheses originating from the participant observation.

Using the principles of experimental design, the analyst assigned specific assessors to particular threats, on which they would *think aloud* and answer the interview questions mentioned above. The design contained four factors and levels within them; two of the factors pertained to distinctions in the threats and two to those in the assessors. For example, one factor pertaining to the threats was their credibility, as determined by the team's original assessments of the threat.[7] This factor had 2 levels—*obviously a hoax* and *not obviously a hoax*—as shown in Table 1. Another factor was the *assessor's stated predisposition to consider a threat not credible*; its levels were *yes* and *no*.

---

[7] That these original assessments were valid is tentatively confirmed by the events following a threat. For instance if a threat was judged not credible and does not occur, the assessment could be considered confirmed.

TABLE 1
Four factors and their levels

factors	levels
threat variety	A, B, C
threat credibility	obvious, not obvious hoax
teams	D, E
predisposition to not credible	yes, no

A general aim in experimental design is for balance because this allows each factor to be evaluated independently of each other factor. Designs most widely used in formal experimentation (e.g., standard factorials) have this property. In the design below, for example, each threat variety is balanced in that each level of each factor is involved in exactly 1/2 of the responses. Moreover, for each combination of factor levels included (*threat variety*, *threat credibility*, *team*, and *predisposition*), its mirror-image combination is excluded.

Ideally, we would have liked all combinations of the factor levels to be observed several times each, in order to compare reproducibility of results for a given combination to overall differences between combinations. However, this was not feasible because of constraints in the number of interviews and the availability of some assessors. Instead, we observed one half of all the possible combinations in a balanced way.

TABLE 2
An example of experimental design

	Threat Credibility					
	Obvious hoax			Not Obvious Hoax		
Threat Variety	Assessor ID#	team	predisp.	ID#	team	predisp
A	1	D	no	7	D	yes
	2	E	yes	8	E	no
B	3	E	no	9	D	no
	4	D	yes	10	E	yes
C	5	D	no	11	D	yes
	6	E	yes	12	E	no

The verbal report and structured interview were administered to the assessors according to the above design. Preliminary analyses were subsequently performed to determine if there was any evidence of such factors as assessors' predisposition to consider a threat not credible or the assessors' serving on a particular team influencing the assessor's conclusions or confidence ratings. The initial analyses indicated that there was no evidence of these factors affecting the assessors' conclusions or confidence ratings. In general, the assessors gave very similar conclusions and confidence ratings to the same threats, particularly when these were threats that had originally been assessed (by the teams) as being obvious hoaxes. The closeness of these conclusions and ratings may have complicated the detection of the subtle patterns, such as the effect of the assessor's predisposition. In addition, because of the small sample size of assessors and the relatively large number of factors, an effect would have to have been pronounced to be

detected; this problem is magnified because responses are categorical instead of being measurable on a continuous scale. We plan to gather additional data and perform further analyses to obtain more definitive results.

4. Guidance on the Use of Participant Observation

As knowledge engineers, our conception of knowledge acquisition, the nature of knowledge, and the goal of our applications influence which methods we select as appropriate. Elicitation techniques, such as participant observation, carry their own assumptions and provide information in a particular form. Shaw and Woodward (1989) have drawn attention to the *patchwork* problem of combining differing knowledge elicitation techniques, typically borrowed from other disciplines. Elicitation techniques make implicit, often conflicting assumptions about the nature of knowledge and its characteristics.

In light of Shaw and Woodward's efforts to promote a methodology for the logical selection of knowledge elicitation techniques, we describe some of the assumptions that participant observation implicitly makes. We hope that the following information will allow the reader to assess the validity of participant observation for specific applications and for combination with other techniques.

In our opinion, participant observation holds few, if any, assumptions on the nature of knowledge. According to Shaw and Woodward (1989, p. 29-2), a technique's implicit assumptions about the nature of knowledge tend to emerge in the knowledge representation language. They cite the example of entity-attribute grids which assume that distinctions are primitives underlying knowledge. In contrast, the representation language of participant observation is generally unstructured text. In this way, participant observation resembles some of the unstructured interviewing techniques which Shaw and Woodward describe (p. 29-15) as making few assumptions about the nature of knowledge.

However, unlike some of these interviewing techniques, participant observation makes few assumptions of how people process knowledge. Interviewing, which usually involves questioning the expert in retrospect about some activity, assumes that the expert is answering the question and that this answer is based on what the expert actually did or thought. These assumptions rest on certain models of what the expert can recall or can bring into short-term memory. Participant observation is not based on these information processing assumptions because the knowledge engineer is not providing the cues or prompts for the expert's responses, the expert's environment is. Furthermore, in participant observation the knowledge engineer is not structuring that environment, such as in giving the expert a particular task to solve.

Implicit within participant observation is the view that knowledge is being constructed by the participants or problem solvers. Participant observation is thought to allow the researcher to look at differences in the participants' behaviors under differing situations as a means of uncovering patterns (which may lead to the discovery of distinctions or rules) in their decision making processes. Thus, participant observation can be considered to assume that participant's actions and interactions are behavioral outputs of their knowledge. Participant observation is further based on the assumption that actions are more reliable indicators of the participants' mental models than their retrospective or even concurrent descriptions of their decision making.

Participant observation is based on the methodological assumption that first-hand information is less prone to distortion than that relayed through other parties. It also assumes that understanding of the participant's views can be accomplished by the researcher's exposure to the same environment and experiences.

In addition, there are a few other considerations in using participant observation.

- As a general rule, participant observation is suited to problem-solving tasks that involve two or more persons in interaction. It can be employed to familiarize the knowledge engineer with the domain or to provide more in-depth information.
- Participant observation not a good choice if the knowledge engineer is likely to have any difficulty obtaining entry into the experts' society. Entry is more difficult if the domain is classified or proprietary, if the groups are opposed to outsiders, or if the groups do not have a concept of what it is to study or conduct research (e.g., ethnologists have frequently entered groups by likening their position to that of students).
- Note that the balance or level of participant versus observer can be adjusted to the application. In some cases, the knowledge engineer may be precluded from participation by nature of task. For instance, if the experts are involved in responding to life-threatening emergencies, the knowledge engineer would not participate because this could imperil the knowledge engineer, the experts, or those to who the experts are responding.
- Participant observation is time consuming and its returns (e.g., in providing insider insight's or in stimulating the knowledge engineer's formulation of hypotheses) may be difficult to measure.
- We would advise the knowledge engineer to expect to be uncomfortable with having a marginal status of not being wholly a researcher, in the traditional empirical sense, nor a participant while doing participant observation.

5. Summary

We have described participation observation to allow others to use it for manual knowledge acquisition. In particular, we believe that participant observation offers the knowledge engineer unique benefits in understanding the expert's domain and in formulating hypotheses on the relationship of the expert's actions to situational factors. We have proposed that the knowledge engineer's hypotheses be tested and then used in constructing knowledge bases, qualitative models of systems in the world. The concurrent verbal protocol, structured interviewing, and statistical experimental design were mentioned as potential techniques for testing the hypotheses. Unfortunately, more conclusive results on the testing of hypotheses must wait for the later stages of this study.

Our aim in this paper was to provide a more explicit, disciplined approach to participant observation. To this end, we have described the uses, strengths, and difficulties of participant observation, the way of thinking that it involves, and its implicit assumptions as to the nature of knowledge and human information processing.

We are indebted to the participants in the study of the threat assessors for their gracious cooperation and donation of their time.

References

BELENKY, M. F., CLINCHY, B. M., GOLDBERGER, N. R., & TARULE, J. M. (1986). *Woman's Ways of Knowing: The Development of Self, Voice, and Mind.* New York, New York: Basic Books.

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

BOOSE, J. H. & SHAW, M. L. G. (1989). Knowledge acquisition for knowledge-based systems, *AAAI-89 Tutorial Notes of the Eleventh International Conference on Artificial Intelligence*, Detroit, Michigan, August.

BOWER G. H. & MORROW, D. G. (1990). Mental models in narrative comprehension, *Science*, 247, 44-48.

BREUKER, J. & WIELINGA, B. (1987). Use of models in the interpretation of verbal data, in A. L. Kidd, Ed., *Knowledge Acquisition for Expert Systems: A Practical Handbook*, pp. 17-42. New York: Plenum 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.

CLANCY, W. J. (1989). Viewing knowledge bases as qualitative models, *IEEE Expert*, 4, 9-24.

CRANE, J. G. & AGROSINO, M. V. (1974). *Field Projects in Anthropology*. Morristown, New Jersey: General Learning Press.

ERICSSON, K. A. & SIMON, H.A. (1980). Verbal reports as data. *Psychological review*, 87, 215-250.

ERICSSON, K. A. & SIMON, H.A. (1984). *Protocol Analysis: Verbal Reports as Data*. Boston, Massachusetts: MIT Press.

GAINES, B. R. (1989). Social and cognitive processes in knowledge acquisition. *Knowledge Acquisition*, 1, 39-58.

HUNTER, D. E & WHITTEN, P., Eds. (1976). *Encyclopedia of Anthropology*, New York, New York: Harper & Row.

KOTTAK, C. P. (1974). *Anthropology: The Exploration of Human Diversity*. New York, New York: Random House.

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. 123-135.

MEYER, M. A., BOOKER, J. M. & BRADSHAW, J. M. (1990). A flexible six-step program for defining and handling bias in knowledge elicitation, in Wielinga et al. Eds., *Current Trends in Knowledge Acquisition*, pp. 237-256. Amsterdam, The Netherlands: IOS Press.

ORTIZ, A. (1969). *The Tewa World: Space, Time, Being, and Becoming in a Pueblo Society*. Chicago, Illinois: U of Chicago Press.

QUILLER, S. (1989). *Color Choices: Making Sense out of Color Theory*. New York, New York, Watson-Guptill.

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.

TIETJEN, G. L (1986). *A Topical Dictionary of Statistics*, New York, New York: Chapman and Hall.

WATERMAN, D. A. (1986). *A Guide to Expert Systems*. Reading, Massachusetts: Addison-Wesley.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

A KNOWLEDGE ACQUISITION SYSTEM FOR ANALYSING PRESCRIPTIVE TEXTS

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

Huge amounts of knowledge are available in human organizations in the form of texts. But, there are still few research projects dealing with the problem of extracting knowledge from texts in order to build knowledge bases. In project A.C.A.T. (Acquisition des connaissances et analyse de textes), we explore the possibility of creating knowledge bases by exploiting information contained in "prescriptive texts" (law, regulations, administrative norms).
We introduce a model for analysing a prescriptive text according to three layers which reflect the juridical metalanguage properties : the macrostructure, the microstructure and the domanial component of the text. We describe the architecture of the knowledge acquisition system SACD and give examples of its use when analysing a regulation text. We present the method which is used to create the knowledge base which is generated by SACD.

1. Introduction

Huge amounts of knowledge are available in human organizations in the form of texts. But, there are still few research projects dealing with the problem of extracting knowledge from texts in order to build knowledge bases. We can put forward various reasons to explain this lack of interest : texts are generally written in natural language (NL) and building a NL understanding system is a very difficult task which is still not fully mastered; only some text types are suitable for knowledge acquisition (for instance, fictions are not very useful for this task); most expert systems have been built as the result of extracting knowledge from domain experts.

However, Regoczei and Plantinga (1988) consider that knowledge acquisition reduces to NL understanding because "words create knowledge by causing a 'cogniting agent' to form new mental models or alter existing mental models". They propose a non-automated method which offers designers some guidelines to identify concepts from the text words and to model their relationships in the form of conceptual graphs (Sowa 1984). In KITTEN (Shaw and Gaines 88) a text analyser "clusters associated words leading to a schema from which the expert can select related elements and initial constructs with which to commence grid elicitation", but the system does not create knowledge structures automatically from the text.

Other researchers have worked on lexical knowledge acquisition from texts. Zernik (1989)

and Jacobs (1988) have developed a system which is used to automatically derive the conceptual category of an unknown word, using pre-existing knowledge on the context in which the word occurred. Velardi et al. (1989) propose a system which analyses a sample of sentences including a given word and produces for that word one or more entries in a semantic dictionary. While these projects are useful for encoding semantic knowledge which can be processed by a NL processing system, they are not advanced enough for elaborating knowledge bases which could be used by expert systems.

In project A.C.A.T. (Acquisition des connaissances et analyse de textes), we explore the possibility of creating knowledge bases by exploiting information contained in "prescriptive texts" which can be found in companies : norm books, regulations, instruction books, etc. Our current work concentrates on French versions of regulation texts from the Government of Québec (Moulin and Rousseau 1989-1, 1989-2, 1990-1). Our approach is based on the following hypothesis :
- some prescriptive texts, such as regulation texts, have a form and a content which can be transformed to create at least the kernel of knowledge bases;
- the statements of a regulation text can be analysed in a systematic way in order to derive knowledge structures which are logically equivalent to the original text and which can be used by an inference engine;
- there is a similarity between the way a text is organized and the way a knowledge base can be structured.

There are multiple sources of juridical knowledge : statute and regulation texts, jurisprudential texts ("case law"), knowledge of legal experts, etc. Research activities have investigated different areas : formulating legislation with the aid of logical models (Sergot et all 1986) (Bench-Capon et al. 1987), legal reasoning (Von der Lieth Gardner 1987), case-based reasoning (Rissland et al. 1987) (MacCrimmon 1989), developing expert systems applied to the juridical (McCarty 1980) (Deschamps 1987) (Nitta 1988) (Schild 1988) (Tidrick 1988) or administrative domains (Corriton 1988).

In the domain of law, most expert systems use "surface knowledge" which are heuristic rules obtained from legal experts ("shallow models"). McCarty (1984) emphasized the need for developing expert systems using "deep knowledge" and Bench-Capon (1989) indicated that this deep knowledge can be partly obtained in formalizing legislation : our research investigates this direction. While analysing a corpus of regulation texts, we identified specific words which are regularly encountered and denote the presence of a metalanguage for expressing juridical statements (Moulin 1988). We have developped a knowledge acquisition system (SACD for "système d'acquisition des connaissances déontiques") which detects within prescriptive texts written in French the juridical metalanguage components and generates a so-called "deontic knowledge base" (DKB).

In section 2 we introduce a model for analysing a prescriptive text according to three layers

which reflect the juridical metalanguage properties. Section 3 describes the knowledge acquisition system. In section 4, we present the method which is used to create the DKB.

2. The component parts of a prescriptive text

2.1 Introduction

We distinguish three layers in a prescriptive text : the macrostructure, the microstructure and the domanial part (Moulin 1988).
The **text macrostructure** is defined as the set of information which are used to organize the text content by adding a "superstructure" which improves the statements' presentation and makes its consultation easier : titles, headings, paragraphs, table of contents, index, references, notes, etc. The macrostructure of a text is usually established according to acknowledged presentation rules (titles, paragraphs, index etc.).

We define the **microstructure** of a text as the set of reserved words, expressions and symbols which are used to structure the content of a prescriptive text and emphasize the logical structure. Here are some examples of words and expressions that are used in French to display the microstructure : "si" *("if")*, "lorsque" *("when")*, "il est interdit de" *("it is forbidden to")*, "il est possible de" *("it is possible to")*, la virgule *(the comma)*, etc. Depending on the nature of the text, the author may use a microstructure formulation that can be more or less systematic. Law and regulation texts follow some rules which are quite systematic and make the microstructure visible enough. That is often the case for norm and procedure texts as well. Cohen (1987) has used a similar approach for analysing the structure of argumentative discourse, while identifying linguistic clues (such as "returning to the previous theme") or connectives (such as "as a result", "in other words" etc.) which are words or expressions used by a speaker to directly indicate the argument structure to the hearer.

We define the **domanial component** of a text as the set of information which characterizes the domain of knowledge and belongs neither to the macrostructure nor to the microstructure.

Analysing the text microstructure enables us to recognize the logical coherence of the line of argument and eventually to identify some inconsistencies in it. So, we can consider the microstructure statements as providing a metalangage used to describe the logical chaining of propositions, according to the particular line of argument (or reasoning) that has been chosen by the author. To study this metalangage we can use modal logic, and, more specifically, deontic logic when it comes to regulation texts (Kalinowski 1972) (Von Wright 1968). According to Kalinowski's suggestions (1972), 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*).

2.2 The text microstructure

Various textual elements support the microstructure of prescriptive texts : modal operators, connectors, inter-statements connectors, statements used to define object categories, metatextual statements.

2.2.1 Operators and connectors

In a regulation text most sentences include a modal operator. We call a **modal operator** any expression which points out the modality in the sentence (Kalinowski 1972). The three most common modalities are :
- Obligation. For example: un chauffeur de taxi effectuant un service de taxi spécialisé **doit** être en possession de son permis de chauffeur de taxi (art.31)
*(a taxi driver who offers a specialized taxi service **must** bear a taxi driver license)*;
- Permission. For example: Une personne **peut** être titulaire de plus d'un permis de chauffeur de taxi (art. 33)
*(a person **may** be the bearer of more than one taxi driver license).*
- Interdiction. For example: Le transport collectif **est interdit** au chauffeur d'une limousine exploitée en vertu d'un permis de taxi (art. 73)
*(group transportation **is forbidden** in a limousine operated according to a taxi license)* ;

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.
For instance in art. 31, the front scope is {un chauffeur de taxi effectuant un service de taxi spécialisé} *{a taxi driver who offers a service with a specialized taxi}*, and, the back scope is {être en possession de son permis de chauffeur de taxi} *{bear a taxi driver license}*.

The modal operator plays an important role within a normative sentence : it helps identifying the object (usually situated in the front scope) on which holds the prescription (usually situated in the back scope); it indicates the modality of the prescription. For instance in art. 31, 33 and 73 the objects are respectively "taxi driver", "person" and "group transportation".

Classical **logical operators** are also identified in the text : "et" *("and")*, "ou" *("or")*, "ne ... pas" *("not")*, "ne ... plus" *("no ... more")*, etc.

A connector si 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.
Consider for instance art. 68 :
"Le titulaire d'un permis de taxi, **lorsqu'**il offre un service de transport collectif, **doit** installer sur la partie avant du toit de l'automobile une enseigne lumineuse".
*(the bearer of a taxi driver license, **when** he offers a service for collective transportation, **must** set up a luminous sign on the front part of the car top)*

We have a connector "lorsqu'" ("when") and its scope is delimited by the comma : {il offre un service de transport collectif} *{he offers a service for collective transportation}.*

Inter-statements connectors characterize a contextual relation between the phrase they introduce and the preceding one. They have no scope. Some inter-statement connectors such as "de plus" *(what's more")*, "en conséquence" *("consequently")* are said to be positive since they introduce a sentence which completes positively the preceding one.
Other inter-statement connectors, such as "cependant"*(however)*, "néanmoins" *(nonetheless)* are said to be negative since they introduce an exception with respect to the context set by the preceding sentence. That's the case in art 45.:

"**Lorsqu'**il reçoit le paiement d'une course, un chauffeur de taxi **doit** remettre la monnaie exacte au passager. **Cependant, si** le paiement d'une course est fait avec un billet excédant de plus de 20.00 $ le prix de la course, le chauffeur de taxi **peut ne pas** accepter le paiement."
*" **When** he receives the payment for a journey, a taxi driver **must** give back to the passenger the exact change. **However**, if the payment is done with a note which goes beyond the fare for more than $ 20.00 , the taxi driver **may not** accept the payment).*

2.2.2 Metatextual statements

We find in regulation texts **metatextual statements** which do not describe application domain properties but properties of the text itself. Here are some examples (X indicates an object category, P holds for a generic proposition, and we use braces to indicate different formulations).

"Les {articles, clauses} C s'appliquent mutatis mutandis {à, aux} X";
{Article, clause} C apply mutatis mutandis to X
This statement specifies that the rules which are part of article or clause C apply as well to the object category X.

"X doit P, conformément à l'article A". *X must P according to article A.*
This statement indicates that the prescription "X must P" applies within the context set by article A.

"{Nonobstant, malgré} les dispositions de l'article A, X doit P".
(Notwithstanding, in spite of) provisions of article A, X must P.
This statement indicates that the prescription "X must P" applies in a context which is contrary to the context set by article A.

Taking in account the application context set by some other text parts, metatextual statements introduce new prescriptions or modify the context within some other part of the regulation (articles, sections etc.). We will say that these metatextual statements express **internal references** within the text.

Some statements define new object categories related to previous categories. For instance a statement like **Tout X qui P, {est, est réputé} Y** , *(every X who P, is said to be a Y)* describes the object category Y as a subclass of object category X.

2.2.3 Anaphoric references

In a natural language text we find terms which are used to refer to objects uttered in another sentence without repeating them. For instance "il" *("he")* is an anaphor which refers to "chauffeur de taxi" *("taxi driver ")* in the following sentence : "Lorsqu'il reçoit le paiement d'une course, un chauffeur de taxi doit remettre la monnaie exacte au passager"*(" When he receives the payment for a journey, a taxi driver must give back to the passenger the exact change").*

Here are some categories of anaphoric references in French.
personal pronouns : il(s), elle(s), le, la, les, lui, leur, l', y, en; *(he, she, it, they, him her, them)*;
adverbs : y, en;
possessive pronouns : le(s) sien(s),. le(s) leur(s) etc.; *(his, hers, its, theirs)*;
possessive adjectives: son, sa, ses, leur(s); *(his, her, its, their)*;
demonstrative adjectives : ce(s), cet(te), tel(s), telle(s); *(this, that, those)*.

These anaphoric references are easily detected in texts. What's difficult is the process of determining to which object refers the anaphoric reference. Solving an anaphor may require knowledge at a semantic or even pragmatic level.

2.5 Our approach for analysing prescriptive texts

As we mentioned in the first section, NL understanding is a very difficult task which is not yet completely mastered, especially when it comes to semantic and pragmatic knowledge embedded in texts. Given the properties of prescriptive texts and for efficiency's sake, we chose to analyse them at a syntactic level, using text grammars. We claim that this kind of analysis is sufficient in order to detect in prescriptive texts the knowledge components

which can be used to structure a knowledge base that will reflect the logical content of the initial text. Hence our approach is domain independent and can be applied for any prescriptive text written in French : we do not need to use large lexicons of the French language because a semantic interpretation of the text is unnecessary.

First, the system detects macrostructure elements in prescriptive texts, using a macrostructure text grammar, and then it identifies in sentences the metalanguage elements using the microstructure grammar. Those grammars are context-free grammars with multiple entry points. For more details on those grammars see (Moulin and Rousseau 1989-1 and 1990-2).
The microstructure grammar enables the system to detect the various elements which support the metalanguage in text sentences : modal operators, connectors, inter-statements connectors and their scopes, as well as internal references. As a consequence of this detection, the system can identify relevant objects of the regulation domain, along with the prescriptions which apply on them (obligation, interdiction, possibility) and the context of their application (composed of the various conditions under which prescriptions must be applied).

As we will see in section 3, after analysing each sentence the system submits it to the domain expert which will accept the parse or correct it. The analysis can be unsuccessful if there is some error in the initial text (mis-spelling, ungrammatical form) or if the text grammars are not complete enough to recognize a new operator or connector. In the first case the domain expert will correct the text. In the latter case the knowledge engineer will update the text grammars in order to process the new case. Currently these grammars are quite stable.

2.4 Structuring the knowledge acquired from prescriptive texts

A regulation text may be considered as a body of knowledge that describes the characteristics of a practical domain on which the law is applied, supposedly in an exhaustive way. From a cognitive point of view we can consider that a prescriptive text describes the properties of a "standard world". The articles that we have presented are excerpts from a regulation text which contains the characteristics of a standard world describing how the transport by taxi (Gouvernement du Québec 1987) has been imagined by legal experts.
This world is composed of abstract or concrete objects, which have been selected by legal experts : for instance "le chauffeur de taxi" *("the taxi driver")*, "l'enseigne lumineuse" *("the luminous sign")*. The regulation articles describe these objects' properties (mandatory, possible, forbidden).
When a user (an analyst in an agency for instance) wants to check a situation or a case against the regulation, he must describe the relevant characteristics of the given situation (called "observation world"). These observed characteristics will be compared with the equivalent characteristics (object properties) in the standard world in order to appreciate

the conformity of the observation world with respect to the regulation.

We propose to structure the knowledge contained in a prescriptive text by identifying the objects of the corresponding standard world. After analysing a corpus of regulation texts, we observed that most relevant objects are situated in the front scope of the modal operators. The corresponding rules of the microstructure grammar have been written in order to detect objects in the front scope of modal operators.
After the microstructure analysis the system presents to the domain expert the various objects which have been detected. If the domain expert decides to include them in the knowledge base, he has to provide the relevant information which will enable the system to classify the objects in a type hierarchy. In addition, the domain expert can include new objects in the type hierarchy at any time. This can happen for instance when processing and solving an anaphor : if the domain expert encounters an anaphoric reference which refers to an object not yet included in the type hierarchy, he can decide to add it in the knowledge base. After this step it is unlikely that a relevant object would not have been included in the knowledge base.

When all text articles have been processed, the system generates an extended form of production rules called "deontic rules" (see section 4). Each rule is associated with an object. Hence the deontic knowledge base which reflects the logical content of the prescriptive text is structured along two dimensions : the object type hierarchy; the associations of deontic rules with the corresponding objects .

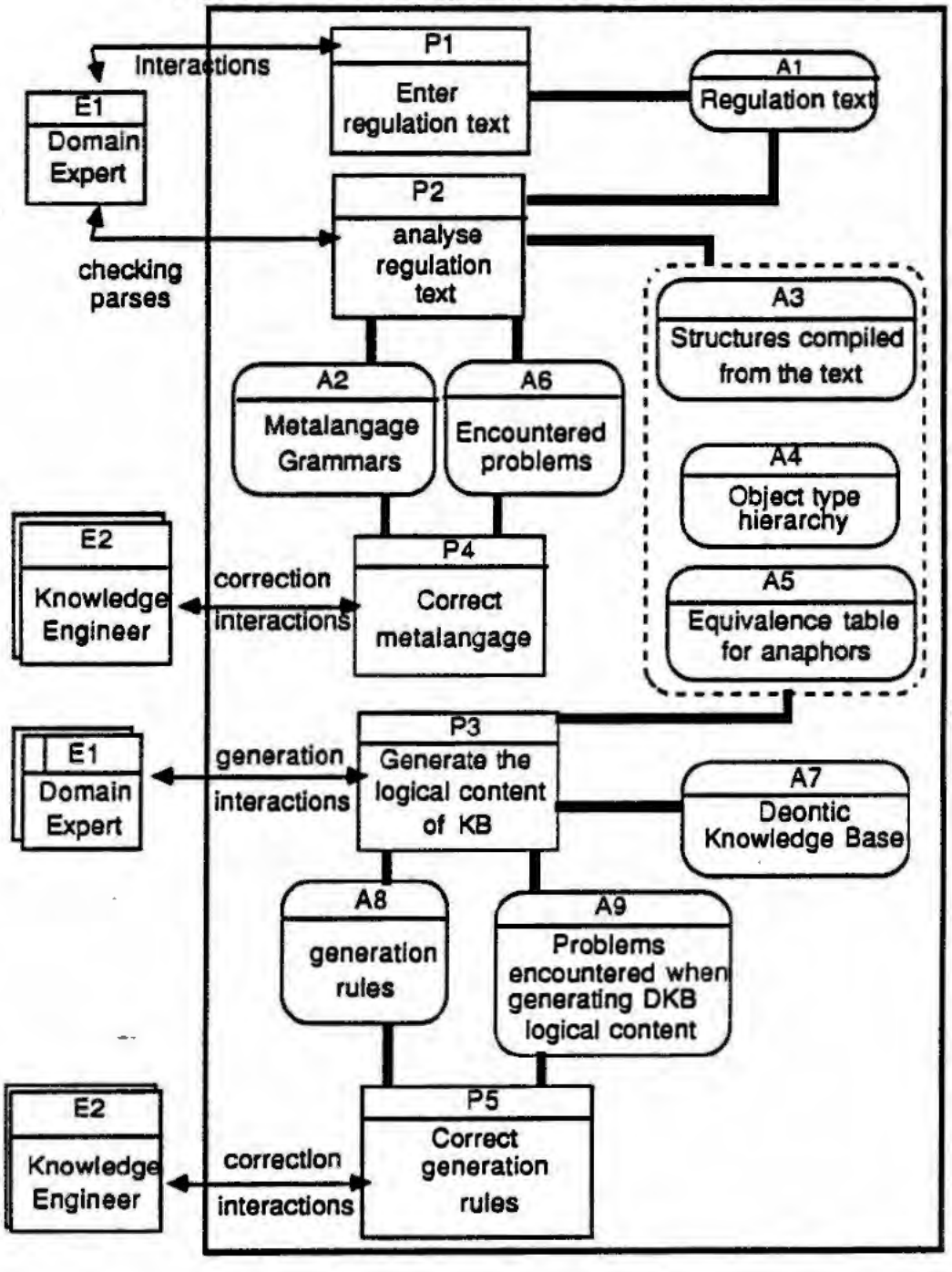
3. The knowledge acquisition system SACD

We propose a knowledge acquisition approach in two main phases : the prescriptive text compilation according to the macrostructure and microstructure grammars; the integration of the compiled structures within the deontic knowledge base. Two types of individuals may use the system : the domain expert (E1) who knows the text characteristics and the intricacies of the legislation; the knowledge engineer (E2) who knows SACD and can modify the grammars if needed. SACD architecture is presented in figure 1.

The first function (P1) enables the domain expert (E1) to enter and edit the regulation text (A1).
Function P2 is controlled by the domain expert (E1) and parses the text (A1) in two steps using the metalangage grammars (macro- et micro- structures text grammars) (A2) and creates the structures compiled from the text (A3). The resulting parses are checked by the domain expert. Then, function P2 interacts with the domain expert (E1) in order to classify new objects in the object type hierarchy (A4). Finally, function P2 presents the detected anaphoric references to the domain expert (E1) in order to solve them and to indicate the referenced objects : an equivalence table for anaphors is updated (A5). If problems are

encountered by the system, they are stored in (A6) and will be solved later by the knowledge engineer (E2). For more details see (Moulin and Rousseau 1990-1).

Figure 1 The knowledge acquisition system SACD



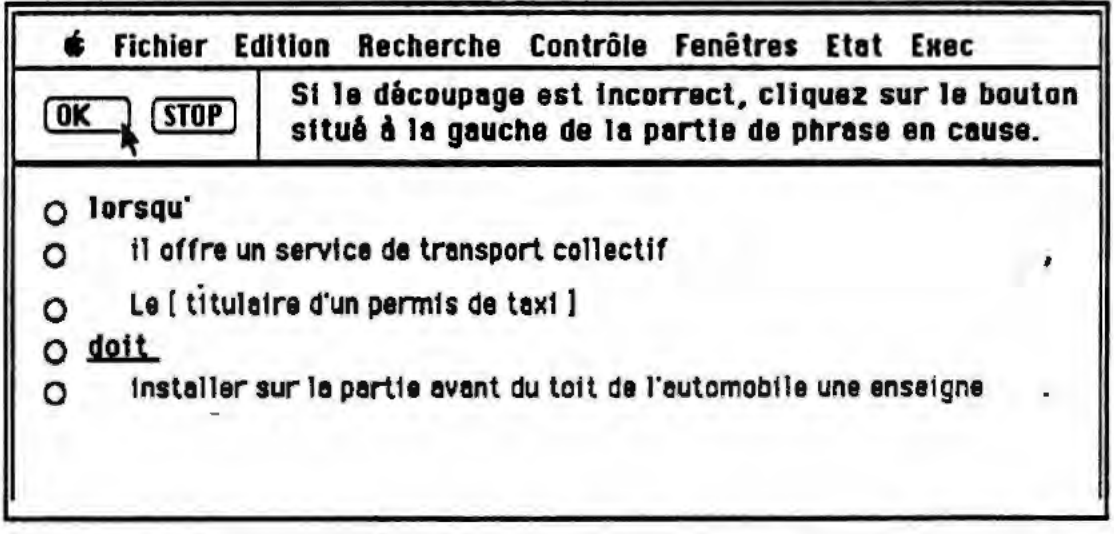
Function P3 proceeds in two steps for generating the logical content of the deontic knowledge

base (DKB) (A7) in the form of extended production rules called "deontic rules". These rules are associated with a modality and attached to the corresponding objects. In a first step, P3 generates the initial content of the DKB while processing the text sentences separately : the deontic rules are obtained along with an initial application context (the rule premisses deriving from the sentence processing). As we have seen in section 2.2, internal references may modify the application context of text articles. In a second step, P3 modifies the DKB logical content, taking in account the internal references which have been detected by function P2 : the application context of some deontic rules may be modified and new rules may be added.

Function P4 enables the knowledge engineer (E2) to correct the metalangage grammars (A2) after studying the problems encountered during the parsing (A6). Hence SACD's text grammars are enriched as the result of the processing of new texts. In a similar way function P5 enables the knowledge engineer to correct the rules (A8) which are used to generate the logical content of the knowledge base (A7) according to the problems encountered (A9) during the generation process.

Let's now illustrate briefly the main steps of text analysis (function P2 in figure 1). For each sentence which has been obtained after the macrostructure analysis, SACD detects the microstructure elements and establishes a parse which is submitted to the domain expert for validation (figure 2).

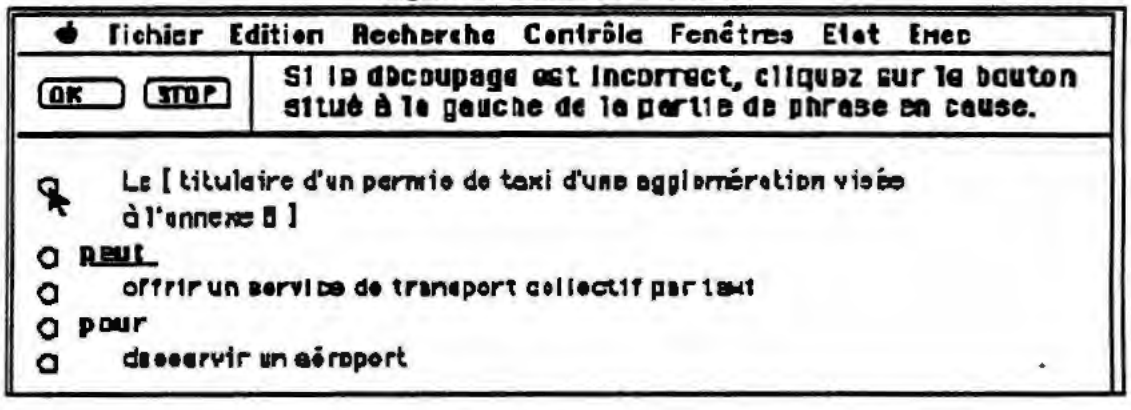
Figure 2 : Parse proposed for article 68



Each microstructure element is presented on a different line. The connector "lorsqu'" *(when)* and the modal operator "doit" *(must)* are presented in a bold style. The scope of connector "lorsqu'" is "Il offre un service de transport collectif" *(he offers a service for collective transportation)*: it is presented on a different line which ends with a comma (","). The front scope of modal operator "doit" is "le [titulaire d'un permis de taxi]" *(the*

[bearer of a taxi driver license]) : it is presented on a different line and the proposed object is presented with square brackets around. The back scope of modal operator "doit" is : "installer sur la partie avant du toit de l'automobile une enseigne"*(set up a luminous sign on the front part of the car top).*
If the domain expert is satisfied with this parse, s/he clicks on the OK button. If not, s/he will click on the button that is in front of the erroneous sentence portion.

Figure 3 : Parse for article 71



Depending on the erroneous element category, different kinds of errors may be detected. SACD asks the domain specialist to indicate the error type, and then enables him to correct the parse.

Let's see for example the parse proposed by SACD for article 71 (figure 3). The domain expert indicates that there is an error in the front scope of the modal operator. SACD presents a menu with different kinds of errors among which the domain expert can choose. Here he chooses an incorrect object (figure 4).

In fact the domain specialist prefers to consider "le titulaire d'un permis de taxi" *(the bearer of a taxi license)* as an object instead of "le titulaire d'un permis de taxi d'une agglomération visée à l'annexe B". SACD asks the domain specialist to put square brackets around the corrected object (figure 5). For more details on error corrections see (Moulin Rousseau 1990-3).

When the analysis of the text microstructure is completed, function P2 displays each object that has been identified in the text and asks the domain expert to specify relevant concepts and class them in the type hierarchy. In figure 6 for instance, SACD suggests to add a new concept "titulaire d'un permis de taxi", and the domain expert clicks on the "ajouter concept" button *("add a concept")* in the menu. The window "objets" enables the domain expert to consult the current list of concepts. Portions of the type hierarchy can be displayed in

another window using the option "dessiner une partie de la grille" *("draw a lattice portion")*in the menu. At any time the domain expert can add a new object in the type hierarchy

Figure 4 : The domain specialist indicates the error type

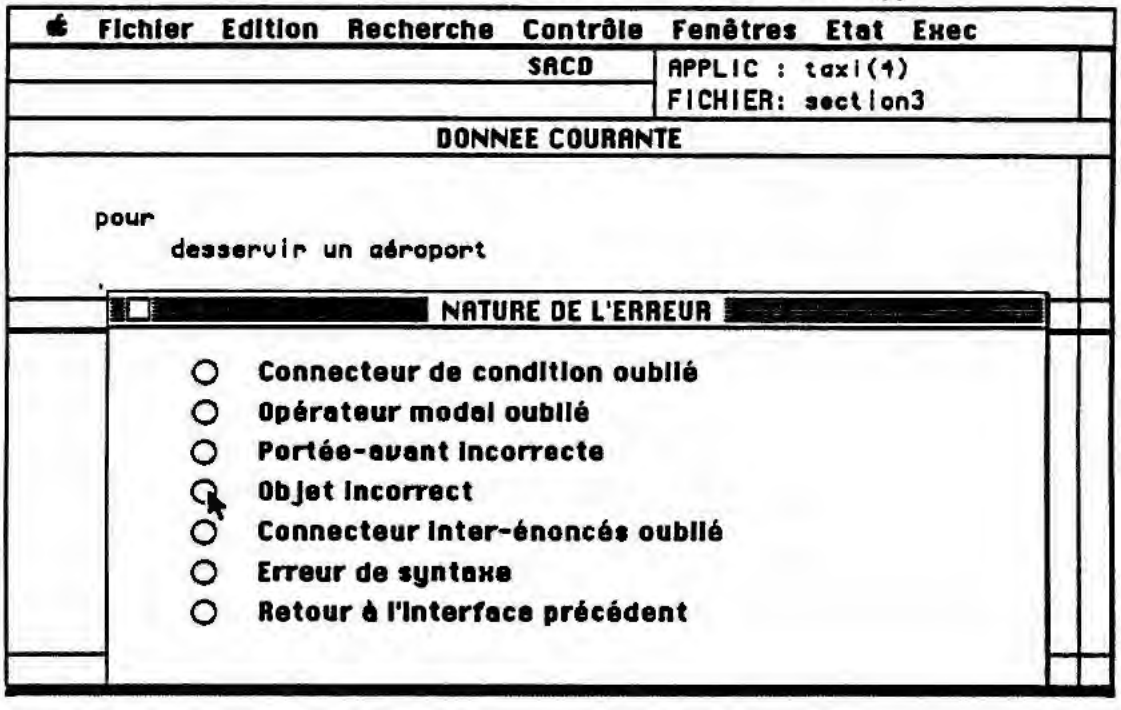
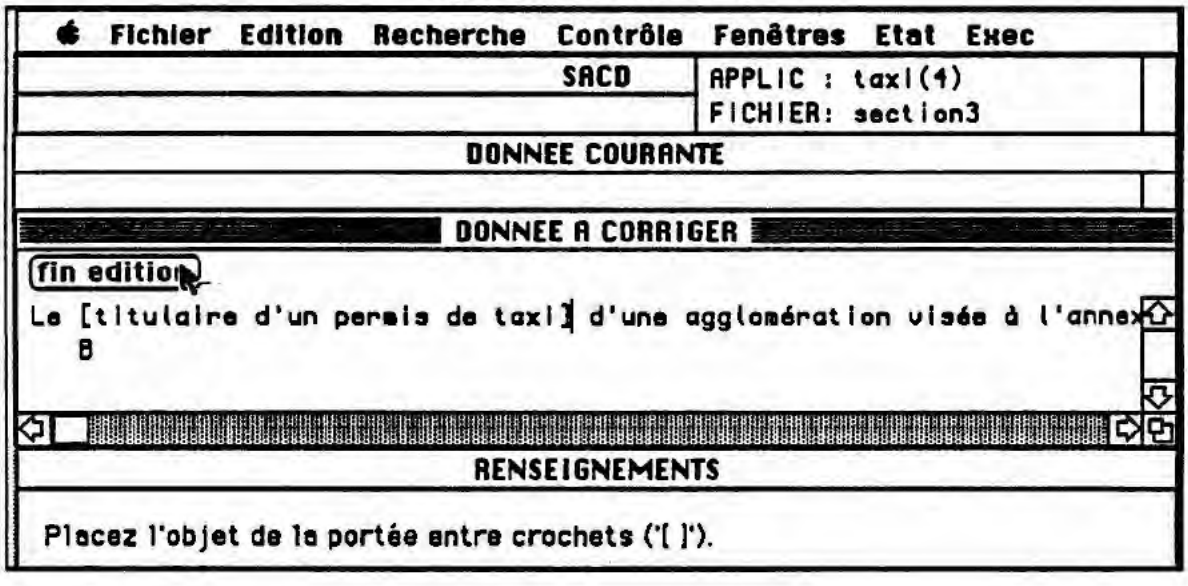


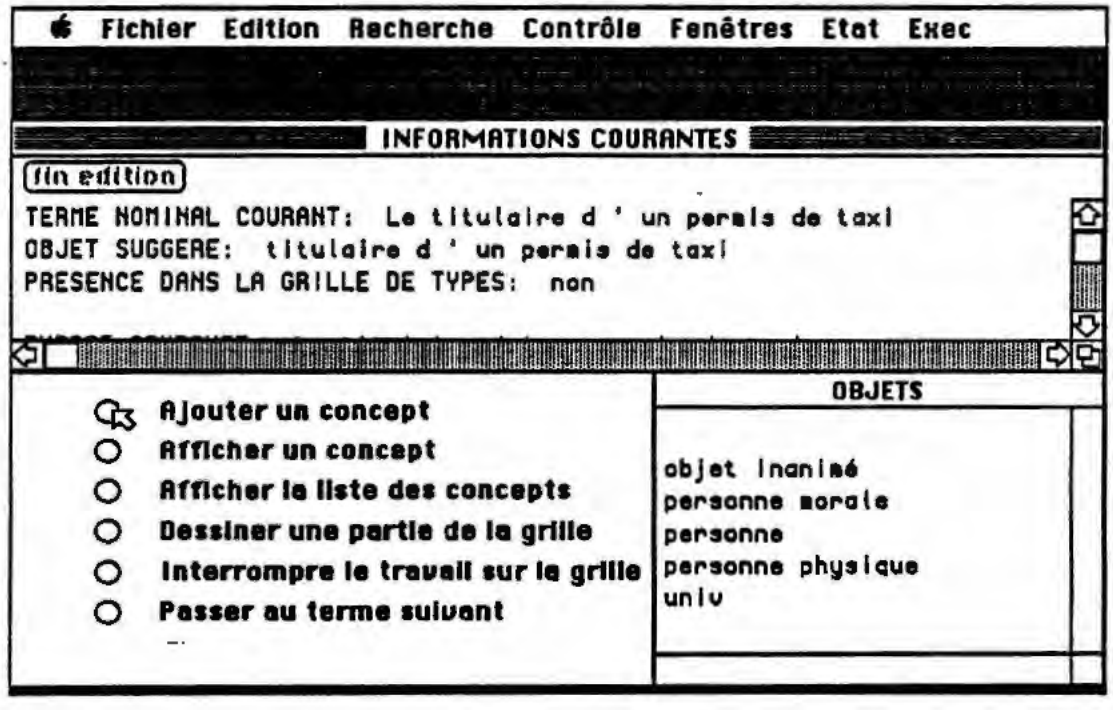
Figure 5 : Error correction



In Figure 7 the window "console" shows the informations attached to the concept "chauffeur de taxi" (taxi driver) : gender, plural form, the lists of super-types and sub-types in the concept hierarchy.

When the objects have been processed and the type hierarchy updated, function P2 asks the domain expert to process the anaphoric references it has detected in the text. Figure 8 shows who the interaction takes place. In the window "extrait de l'article" SACD displays the text portion in which it has detected anaphoric references : they are identified by a number in parenthesis. For instance in sentence "lorsqu'il (1) cesse d'offrir un tel(2) service, il(3) doit l'enlever";*"when he(1) stops offering such a (2) service, he(3) must remove it"*. The dialogue window asks the domain expert to indicate for each anaphor to which object it refers. Here "il(1)" refers to "chauffeur de taxi" *("taxi driver")*. In window "texte" the entire text can be consulted and in window "objets" the current list of objects can be examined.

Figure 6 : SACD suggests to add a new concept "titulaire d'un permis de taxi"



It can be noted that SACD has not identified "l'" *("it")* as an anaphoric reference. In French text personal pronouns "le, la, l', les" are the same terms which are used as definite articles. In order to ease the interaction, we chose not to display them and advise the domain expert to add such references, when he detects them.

Figure 7 : Information describing the concept "chauffeur de taxi"
(taxi driver)

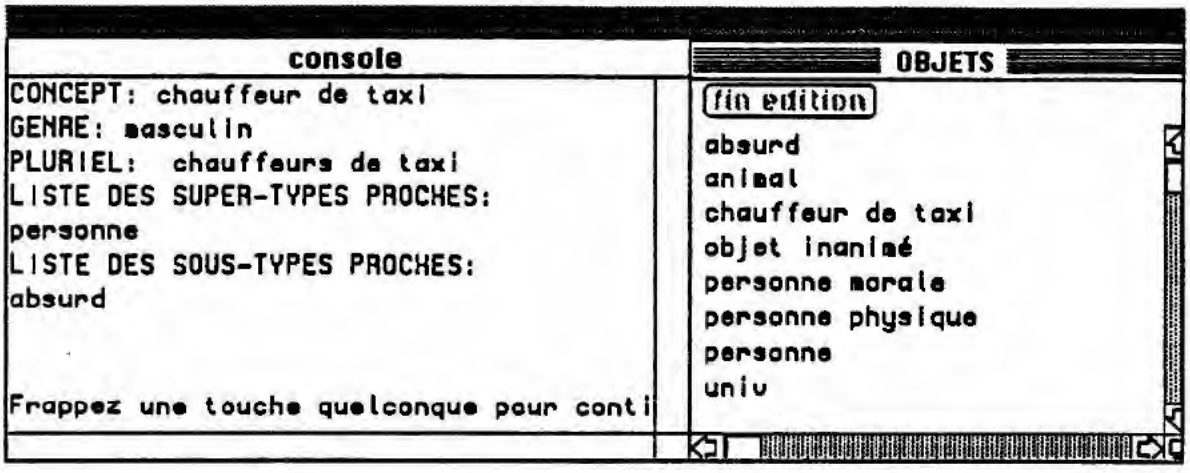
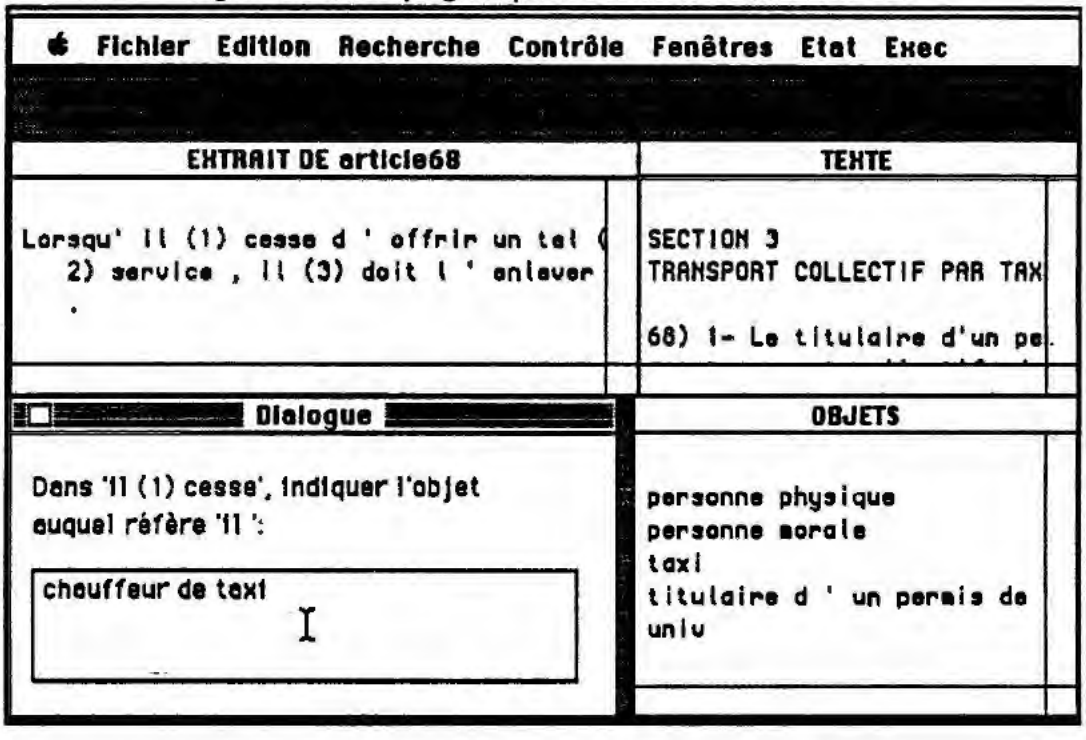


Figure 8 : Identifying anaphoric references in article 68



Resolving anaphoric references ends text analysis (function P2). SACD has stored in the text compiled form all extracted information from which it will be able to generate the deontic knowledge base (DKB).

4. Structuring the knowledge base

Process P3 (figure1) enables SACD to generate the DKB logical content from information stored in the text compiled form. Classical production rules cannot be used to structure the DKB, because we must specify modalities within the rules. The DKB logical content is studied according to a modal logic variant called deontic logic (Kalinowski 1972, Hilpinen 1981, von Wright 1968). As we wanted an inference engine to use the DKB in a consultation mode, we selected the following principles :

- the DKB is composed of so called "deontic" rules which are associated with a modality and attached to the corresponding objects;
- rules are independant of each other;
- conditions and/or exceptions which are encountered in the text sentence are used to generate the base context of the corresponding deontic rule;
- the base context of the corresponding deontic rule can be enriched if other sentences contain internal references pointing to sentences associated with this rule.

A deontic rule v-rule that is associated with a phrase v-phrase and the corresponding article v-article, is composed of a "deontic conclusion" v-deont-concl and an application context v-context :

<v-rule, v-phrase, v-article, v-deont-concl, v-context> -> ;

In the "deontic conclusion" v-deont-concl, we characterize the object part v-object-concl, the modal operator category v-log-op, and the proposition v-back-scope which corresponds to the modal operator back scope in the associated text sentence (see section 2). In the object part v-object-concl which corresponds to the modal operator front scope, we differentiate the object v-object from its determiner and modifier.

<v-deont-concl, v-object-concl, v-log-op, v-back-scope> ->;
<v-object-concl, v-front-scope> ->;
<v-front-scope, v-determiner, v-object, v-modifier> ->;
v-log-op -> OBL ! POS ! INT ! NOB ! NPO ! NIN ! ASS ; where

OBL (obligation), POS (possibility), INT (interdiction), NOB (negated obligation), NPO (negated possibility), NIN (negated interdiction), ASS (simple assertion).

In the remainder of this paper we will use a simplifiad formulation for deontic rules :

v-rule : SI v-contexte < v-obj-concl, v-log-op> v-deont-concl.

The reader will find some examples of deontic rules in section 5.

The application context **<v-context>** in the deontic rule is a list of couples
<v-premises, truth-val> and/or a list of premises disjunctions
<disj-premises> (simple premises linked by 'AND' operators).

A premises disjunction <disj-premises> is a list of couples
<v-premise, truth-val> (simple premises linked by 'OR' operators).

The truth value truth-val specifies if the application context is positive (truth-val = 1) or negative (truth-val = 0).

Process P3 (figure 1) generates the DKB logical content in two steps:
- **Step 1** : for each phrase (v-phrase) found in the prescriptive text, SACD generates a deontic rule (v-rule) with its base 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. Inter-statements connectors are processed at this level as well.

5. Processing internal references

It is interesting to study the influence of internal references on the DKB logical content. Some of them have no influence; they correspond to metastatements such as "dans le présent réglement" *("in this regulation")*, "le cas échéant"*("should the occasion arise")* etc.
We have identified four other categories of internal references. For each of them we give here a typical example (for more details see Moulin and Rousseau 1990-1).
Cat1 : **{malgré, sous réserve de} l'article** a, X **doit** P **[si** contexte1**]**
{In spite of, subject to} article a, X **must** P **[if** contexte1**]**
Cat2 : **{le présent article, la présente section} [ne] s'applique [pas] aux** X, Y et Z
This {article, section} {don't apply, applies} to X, Y and Z
Cat3 : in a sentence part we can find : un X **{conforme, mentionné} à l'article** a
an X **{in conformity with, mentionned in} article** a
Cat4 : this category corresponds to inter-statements connectors (see section 2.2.1)

As an illustration we present how SACD processes an internal reference introduced by "malgré"(cat1) in a sentence whose pattern is :

Ph x : **Malgré l'article** a, X **log-op**$_x$ P_x **si** contexte$_x$
In spite of article a, X **log-op**$_x$ P_x **if** contexte$_x$

During the first step of function P3, a rule has been generated for article a :

R_a : IF v-contexte$_a$ < v-obj-concl$_a$, v-log-op$_a$> v-deont-concl$_a$

During the second step of function P3, the internal reference introduced by "malgré" is processed:

- R_a is modified :
  IF v-contexte$_a$ **AND NOT** contexte$_x$< v-obj-concl$_a$, v-log-op$_a$> v-deont-concl$_a$
- a rule is created and associated with phrase Ph_x:
  v-rule$_x$: **IF** v-contexte$_x$ < X ,op-mod$_x$> P_x

Consider article 63 : un [chauffeur de taxi] **ne 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 *(conclusion 63)* **à moins que** la destination de cette course ne soit située à plus de 50 kilomètres des limites de cette agglomération ou de cette région *(contexte 63)*.
A [taxi driver] **cannot** refuse to run an errand whose origin is situated within an urban center or region for which the taxi license has been issued *(conclusion 63)* **unless that** the errand destination is more than 50 kilometers away from the limits of this urban center or region *(context 63)*.

During step 1 of function P3 the following rule is generated :

R63 : **IF NOT** contexte63 « chauffeur de taxi, NPO» conclusion63

Note that "IF NOT" appears in R63 context because of connector "unless that".

Consider article 64 : **Malgré l'article 63**, un [chauffeur de taxi], **s'il n'occupe pas** la première place à un poste d'attente *(contexte64)*, **doit** refuser ses services à un client qui s'y présente directement et lui signifier la possibilité de faire sa réquisition au premier taxi en attente*(conclusion64)*.

In spite of article 63, a [taxi driver], **if he is not** in first row at a waiting post *(context 64)*, **must** refuse to serve a customer who wants to embark with him and tell him that he can ask to the first taxi waiting at the post*(conclusion64)*.

Note that context 64 corresponds to the positive form of the proposition :"il occupe la première place à un poste d'attente" *("he is in first row at a waiting post")*. When processing article 64, SACD

- modifies rule R63 :

R63 : **IF NOT** context63 **AND** contexte64 « chauffeur de taxi, NPO» conclusion63

- creates rule R64 :

R64 : **IF NOT** contexte64 « chauffeur de taxi, OBL» conclusion64.

6. Conclusion

SACD is written in Prolog II and runs on MacIntosh workstations. It has been successfully used to generate knowledge bases from various regulation texts of the Government of Québec (from ministries for transportation, communications and environment). As we said earlier, the macrostructure and microstructure text grammars are now fairly stable. If new operators or connectors are found in a new text that SACD processes, the knowledge engineer analyses the case and updates the grammars.
The macrostructure analysis of a 150 articles regulation text takes around 5 minutes, but no interaction with the specialist is needed, unless SACD finds some inconsistencies in the text presentation. The microstructure analysis of a sentence within an article depends on the complexity of the sentence structure, but it is usually completed in less than 5 seconds. The DKB can be used for various purposes. We are currently working on a system which will check the knowledge base consistency. If some inconsistency is detected it will be presented to

the specialist with the corresponding text articles, in order to correct them. The most common inconsistencies arise from a misuse of internal references in the text.

Our research project extends in two other directions.
We are working on the development of a consultation system which will interact with a user in order to check cases against the regulation. The consultation will use the DKB to ask questions to the user and to help him while describing the case. Then a diagnosis will be issued by the system about the case conformity compared with the legislation. The consultation system will have to phrase its questions in French.

We are also working on the development of a system which will help domain experts to design and write prescriptive texts, using the knowledge which is stored in SACD grammars. Up to now the only tools that are used by domain experts in organizations are word processors and this lack of support makes the writing process very lengthy and costly, and texts are far from being perfect. We are currently working on a method which enables domain experts to use some kind of semantic networks to model the application domain. The next phase will enable us to propose a tool to support them.

ACKNOWLEDGEMENTS
This project is supported by the CEFRIO, le Centre francophone de recherche en informatisation des organisations (subvention 1987-90).
We would like to acknowledge all the people from various governmental institutions for their help and encouragements during our "navigation" in the administrative and legal documents.

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).
BENCH-CAPON T. J. M. (1989), Deep models, normative reasoning and legal expert systems, in (ACM 1989).
COHEN R. (1987), Analyzing the structure of argumentative discourse, in *Computational Linguistics*, vol 13 n.1-2, pp 11-24.
CORRITON J. C. (1988), F. M. Lesaffre, SYMA : système expert en milieu administratif, in (RAULT 1988).
DESCHAMPS J. (1987), M. Quenillet, Systèmes experts juridiques : une réalité, in proceedings of the seventh Workshop Expert Systems and their Applications, Avignon May 1987.
GOUVERNEMENT DU QUEBEC (1987), Règlement sur le transport par taxi, révisé 1987.
HILPINEN R. editor (1981), *New Studies in Deontic Logic*, D. Reidel, Dordrecht, 1981.

JACOBS P. (1988), ZERNIK U., Acquiring lexical knowledge from text : a case study, Proccedings of the AAAI88, St. Paul, Min.

KALINOWSKI G. (1972), *La logique des normes*, Presses Universitaires de France.

LEE (1988) R. M., Bureaucraties as Deontic Systems, *ACM Transactions on Office Information Systems*, vol 6 n 2, April 1988.

MacCRIMMON M. T. (1989), Expert systems in Case-based law : the Hearsey rule advisor, in (ACM 1989).

McCARTY J. (1980), N. S. Sridharan, The representation of an evolving system to legal concepts : Logical Templates, in proceedings of the Third Biennal Conference of the Canadian Society for Computational Studies of Intelligence, Victoria, B. C. 1980.

McCARTY J. (1983), Permissions and obligations, in proceedings of the Eighth IJCAI.

McCARTY J. (1984), Intelligent legal information systems : problems and prospects, in *Data Processing and Law*, C. Campbell editor, Sweet and Maxwell pub. comp.

MOULIN B. (1988), Réflexions sur la conception et l'utilisation de bases de connaissances déontiques, Rapport de recherche CEFRIO.

MOULIN B. (1989-1), 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. (1989-2), D. Rousseau, Analyse de textes prescriptifs pour la génération de bases de connaissances déontiques, actes du colloque international ICO'89, publié dans B. Moulin, G. Simian éditeurs, *L'informatique cognitive des organisations*, L'interdisciplinaire, France.

MOULIN B. (1990-1), D. Rousseau, Structuration d'une base de connaissances à partir du contenu de textes prescriptifs, proceedings of the Eighth Workshop on Expert Systems and their Applications, Avignon France May 1990.

MOULIN B. (1990-2), D. Rousseau, Designing deontic knowledge bases from regulation texts, to appear in the Journal *Knowledged-Based Systems*.

MOULIN B. (1990-3), D. Rousseau, un outil pour l'acquisition des connaissances à partir de textes prescriptifs, à paraître dans la revue *ICO Québec*.

NITTA K. (1988), J. Nagao, T. Mizutori, A Knowledge Representation and Inference System for Procedural Law, *New Generation Computing*, vol. 5 n. 4.

RAULT J. C. editor (1988), proceedings of the Eighth Workshop on Expert Systems and their Applications, Avignon France May 1988.

REGOCZEI S. (1988), PLANTINGA E. P. O., Creating the domain of discourse : ontology and inventory, in *Knowledge Acquisition Tools for Expert Systems*, J. Boose and B. Gaines editors, Academic Press.

RISSLAND E. L. (1987), K. D. Ashley, A case-based system for trade secrets law, in (ACM 1987).

SHAW M. L. G. (1988), GAINES B. R., KITTEN : Knowledge initiation and transfer tools for experts and novices, in *Knowledge Acquisition Tools for Expert Systems*, J. Boose and B. Gaines editors, Academic Press.

SCHILD U. J. (1988), JURIX : a legal expert system, in (RAULT 1988).

SERGOT M. J. (1986), F. Sadri, R. A. Kowalski, F. Kriwaczek, P. Hammond, H.T. Cory, The British Nationality Act as a logic program, Communications of the ACM 29.

SOWA J.F. (1984), " *Conceptual Structures : Information Processing in Mind and Machine*", Addison Wesley.

TIDRICK T.H. (1988), Expert consultant for law enforcement : the case of alleged revenue officer embezzlement , in (RAULT 1988).

TREMBLAY (1984) R. Tremblay, 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.

VON DER LIETH GARDNER A. (1987), *An artificial Intelligence Approach to Legal Reasoning*, The MIT Press, Cambridge, Mass.

VON WRIGHT G. H., (1968) An essay in deontic logic and the general theory of action, in *Acta Philosophica Fennica*, vol 21, North Holland, Amsterdam 1968.

ZERNIK U. (1989), Lexicon acquisition : Learning from corpus by capitalizing on lexical categories, in proceedings of IJCAI89, Detroit, Mich.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Verification of Frame and Semantic Network Knowledge Bases

Daniel E. O'Leary

Graduate School of Business
University of Southern California
Los Angeles, California 90089-1421
213-743-4095

ABSTRACT

Virtually all previous research in verification of knowledge bases has concentrated on rule-based systems. Since most systems and shells developed to-date have concentrated on knowledge representation using rules, this seems appropriate. However, increasingly, there are systems being built using other forms of knowledge representation, such as frames and semantic nets. As a result, there is a need to develop verification approaches for these types of knowledge representations. Verification of frames and semantic networks is investigated from a domain independent approach that focuses on the characteristics of the knowledge representation. Examples are used to illustrate each approach. Implementation considerations also are discussed.

1. INTRODUCTION

Verification was defined by Adrion et al. (1982) as "the demonstration of the consistency, completeness and correctness of the software." The purpose of this paper is to develop approaches to the verification of knowledge represented as either frames or semantic networks.

In knowledge-based systems, many verification efforts are aimed directly at verifying the knowledge base portion of the software. This results from both the design and development processes, and the software used with expert systems. Typically, from a design and development perspective, the knowledge base is treated as an independent entity. In addition, the existence of expert systems shells allows the user to assume that the inference engine produces appropriate inferences, so that the focus can be on the development and verification of a knowledge base.

If assumptions are made about the knowledge base (e.g., assume it is in the form of rules), then the pursuit of consistency, completeness and correctness take greater structure. The more restrictive the assumptions on the knowledge representation or domain, the more structured the verification efforts can be made. Thus, verification can be, to a large extent, a function of specific knowledge

representation technology on which the system is based or the domain in which the system resides.

For example, there has been substantial research on verification of rule-based systems. Those efforts have focused on the structure of the knowledge contained in the rules (Nazareth, 1988 and Nguyen et al., 1985 and 1987) and the structure of the weights on those rules (O'Leary, 1990).

1.1 Use of Frames and Semantic Networks

Although most of the expert or knowledge-based systems that have been developed likely are rule-based, increasingly systems are being developed using alternative knowledge representation schemes. For example, frames and semantic networks (nets) are receiving greater use in many application systems (e.g., Willingham and Ribar, 1988). In part this is because the systems that are being built are more complex, so that alternative forms of knowledge representation are necessary. Since systems increasingly employ other forms of knowledge representation, there is interest in establishing verification of other types of knowledge-based systems.

Since most systems that employ frames or semantic networks do not use shells that embed verification efforts, the verification often is the hands of the developer. In addition, as greater effort is made to develop shells that employ alternative knowledge representations there is a need to incorporate verification devices in those shells. Unfortunately, there is little literature established on the verification of frames or semantic networks. Thus, the purpose of this paper is to elicit approaches to assist in the verification of frame and semantic network knowledge bases.

1.2 Domain Dependent and Independent Approaches

There are at least two ways to characterize the verification process: domain dependent and domain independent (e.g., Nazareth, 1989). This paper is aimed primarily at domain independent approaches.

Domain dependent verification attempts to exploit what is known about the domain, in the form of meta-knowledge, to assist in the verification process. Such meta-knowledge is either embedded in the system or accessed by exploiting human's knowledge of the domain. Some of the first verification approaches were of this nature (Davis, 1976).

Domain independent verification uses knowledge about the particular form of knowledge representation to assist in the verification process, but does not employ any knowledge about the domain. For example, in the case of rules, typically it

is inappropriate to have cycles in the knowledge base. Domain independent approaches use this knowledge about the knowledge representation to assist in the verification process.

Outline of this Paper

This paper proceeds as follows. Section 2 summarizes some of the characteristics of frames and semantic nets and provides definitions used later in the paper. Section 3 investigates domain dependent verification of frames and semantic networks. Section 4 analyzes domain independent verification of frames and semantic networks. Section 6 provides a brief summary of the paper.

2. FRAMES AND SEMANTIC NETS

Although rules probably are the most frequently used form of knowledge representation, frames and semantic networks are among the most used of other forms of knowledge representation. This section provides a brief summary of some of the primary characteristics of frames and semantic nets.

2.1 Frames

As noted by Minsky, the developer of the theory of frames,
in 1975,

> A frame is a data-structure for representing a stereotyped situation like being in a certain kind of living room or going to a child's birthday. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these confirmations are not confirmed.
>
> We can think of a frame as a network of nodes and relations. The 'top' levels of a frame are fixed and represent things that are always true about the supposed situation. The lower levels have many terminals -- 'slots' that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. (The assignments themselves are usually smaller 'subframes.') Simple conditions are specified by markers that might require terminal assignment to be a person, an object of sufficient value, or a pointer to a subframe of a certain type. More complex conditions can specify relations among the things assigned to several terminals.

As an example of a typical frame, Winston (1979) (See figure 1) displays a frame that includes title information of the frame, action information, actor information, object information, source information, destination information, results storage (that lead to a state change frame) and subprocesses information. In each case, except the title, there can be a number of slots to contain information about, e.g., subprocesses. As a result, slots may have considerable information attached to them.

Frames also are hierarchical, so that diagrams of frame-based systems take on a tree-like structure (figure 2) or an acyclic network structure (figure 3). At each level of the tree the frames are likely to be of the same basic design.

Structurally, the "top" frame can be referred to as the root frame. While, the "bottom" frames can be referred to as the terminal frames. Frames between the top and bottom frames can be referred to as intermediate frames. In figure 2, the root frame is the room frame and the terminal frames are the picture frame, the window frame and the door frame.

In some cases the frames lead to other frames, as in figure 1. In some knowledge bases those frames establish parallel structures. For example, in figure 3 each of the group frames leads to two object frames, which in turn lead to an object frame.

2.2 Semantic Networks

Originally semantic nets were designed as a representation structure of the meanings of words. In semantic networks, knowledge is represented as a set of nodes and directed arcs, where the arcs are used to represent relationships between the nodes and the nodes represent objects and characteristics of those objects. Example semantic nets are located in figures 4 and 5.

In semantic networks, there are three types of nodes: root nodes (no arcs leading in -- only arcs leading out); intermediate nodes (must arcs leading in and out); and terminal nodes (only arcs leading in -- no arcs leading out). In some cases there may be little or no parallel structure (figure 4), while in other cases there may be substantial parallel structure (figure 5). Structurally, semantic nets are either trees (figure 4) or acyclic networks (figure 5). Network forms of semantic nets would seldom, if ever, be designed to allow a cycle to occur. Further, in some cases, it may be possible to set up an a priori precedence among types of arcs that would occur. For example, it may be possible to state that an "isa" arc always proceeds a "color" arc. In the examples, no such relationships are established.

3. DOMAIN DEPENDENT VERIFICATION

Domain dependent approaches assume that there is some meta- knowledge available that allows the system to examine resulting knowledge for correctness, completeness and consistency. For example, in the case of figure 2, such domain knowledge could include feasible information for different rooms in a house.

Consider the case of living rooms. If some slot information were to include a stove or a toilet then the verification should ascertain that there was an error, since these are seldom found in living rooms. Similarly, if the "picture" frame (or a "picture frame") was empty then verification could suggest that there was an error of omission, since most living rooms have pictures in them. As noted by Nazareth (1989, p. 263), "verification is not performed, per se, but is achieved through debugging of erroneous system recommendations."

3.1 Integration with Knowledge Acquisition

Since domain dependent verification employs knowledge about the domain, it can be treated as a part of the knowledge acquisition process. Domain dependent approaches can be embedded in the knowledge acquisition processes and tools that support knowledge acquisition. In addition, as these systems accumulate knowledge about the domain, they can use that knowledge to verify new knowledge solicited as part of the process.

For example, consider the case of automated knowledge acquisition, where the expert uses the system and the system assists in the acquisition of that knowledge. For verification purposes, the user can function as one source of the meta-knowledge or the meta-knowledge can be embedded in the system or both. Further, the knowledge acquisition approach may be generic or domain dependent.

Broad based opportunities exist for the use of domain knowledge in verification, including domain knowledge, problem type knowledge (diagnostic), promulgated problem solution methods (smart questionnaires), etc. Virtually any knowledge or decision structure that has a known structure can be used to determine if the ascertained structure is similar to the expected structure. These choices are illustrated in figure 6.

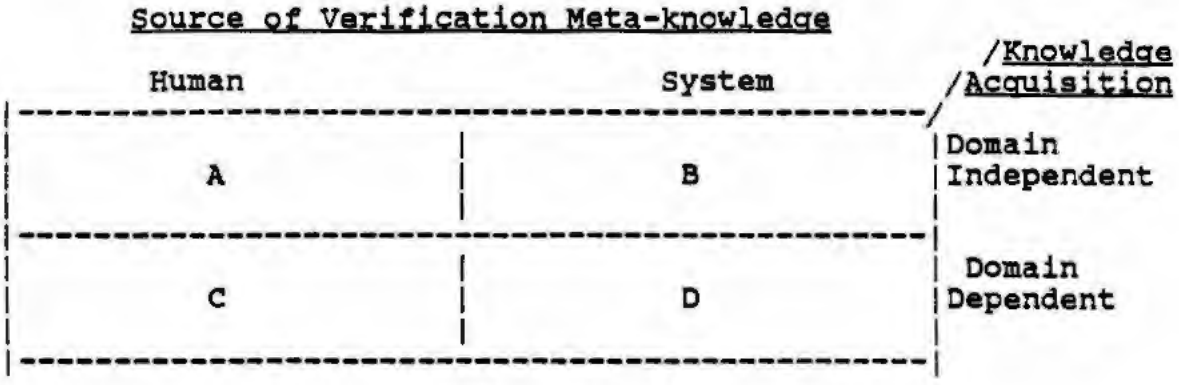


Figure 6

Meta-knowledge and Verification

A comprehensive summary of many different machine-based knowledge acquisition approaches is provided in Boose (1989). Most of those systems are generic, so that they can be used for many different domains. For example, AQUINAS and KITTEN use hierarchically-structured repertory grid interviewing. Most of those systems would thus fall into either category A or B. Other systems such as ID3 are machine learning-based systems, and would fall in category B.

In both types of systems (A and B), the underlying methodological approach could be used in the verification process. For example, in the case of ID3, if the system is unable to classify each of the cases then that can indicate inconsistency in the data and knowledge. Similar constraints resulting from problem methodology can be developed from other approaches, such as decision trees.

A few of the systems discussed by Boose are domain dependent, falling into categories C and D. For example, LEAP uses apprenticeship learning to learn steps in VLSI design, MUM is concerned with medical problems and STUDENT acquires knowledge for the statistical consulting domain. These systems contain meta-knowledge that potentially could be useful for domain dependent verification. In addition, some systems make assumptions as to the type of problem to be solved (e.g., diagnostic problems), thus, providing additional structure for verification.

In any of A through D, the system can be adapted to verify based on the knowledge acquired, either by constantly relating it back to the newly acquired knowledge or by comparing to existing knowledge. In such comparisons either

the user's or the system's domain knowledge can provide the basis of comparison. If the knowledge is there -- use it.

3.2 Limitations

Domain dependent approaches have some limitations. First, with a domain dependent approach, the meta-knowledge of such a system also requires verification. Even in the case of human users, potential problems such as inconsistency can allow errors to permeate the system. Second, if the meta knowledge required is not stable, then the verification portion of the system could require frequent updating. This could lead to a substantial system cost. Such, domain dependent approaches are likely to be particularly useful in large or ongoing systems, where such meta knowledge may be stable and where such a system could receive frequent use. These factors could mitigate system costs. Third, even though the system may have domain expertise, compared to the human expert the system's knowledge may be shallow. As a result, domain dependent approaches may provide limited return.

4. DOMAIN INDEPENDENT VERIFICATION

Domain independent verification uses knowledge about frames and semantic nets, as forms of knowledge representation in general, to assist in the verification process. Accordingly, the purpose of this section is to elicit that kind of knowledge that can be helpful in verification efforts.

It is important to note that the tests generated to verify a system are not guarantees that the system is one hundred percent correct. Instead, they become ways to determine when there are ascertainable errors of the type delineated here.

4.1 Consistency

Consistency refers, in part, to the names or labels used in the knowledge representation. Since humans have a tendency to make errors (e.g., in typing) or call the same thing by different names, procedures need to be adopted to ensure that the names and and contents are consistent. Consistency also can refer to consistent implementation of parallel structures.

Frames. In frames, consistency is a concern with the names given to the frames, frame slots and contents. One approach to eliminate errors deriving from inconsistency is to require the development of different lists for frame names, slot names and slot contents. System construction and use of frames would then require all choices to be from those lists. This would eliminate errors caused by alternative names,

spellings, etc. Thus, part of the knowledge acquisition process is settling on such details.

For example, as noted in figure 2, the frame name is "room frame." This approach could require the construction of a frame by that name only if the frame name were given before the construction of the frame. In addition, the system would constrain use of that name to a single frame. Further, construction of, or reference to a frame would require choosing that frame's name from a list.

Consistency also refers to parallel treatment of parallel structures. If the knowledge base employs a set of parallel structures, as illustrated in figure 3, then the system can request that the user elicit parallelism structures prior to entering knowledge bases. When the system is aware of those parallel structures, their correctness can be monitored.

Allowing parallel structures also can facilitate capture of knowledge by the system. Rather than requiring a piece-by-piece structuring of the network of frames, allowing copying of existing parts of the network make the job of knowledge elicitation easier.

Semantic Networks. In semantic nets, consistency is an issue in the names given to arcs and nodes, e.g., "isa" and "chair." In order to eliminate multiple names being assigned to the same object, e.g., rather than calling a "chair" a "stool," the system can require that a list of eligible node and arc names be developed before those names are entered into the knowledge base. As in the case of the frames, construction of the network would require use of a name from one of those lists. Further, in a manner similar to that discussed for frames, parallel structure can be accounted for by the system when it is aware of its existence.

4.2 Redundancy

Redundancy refers to the introduction of potential undesirable duplication of knowledge into the knowledge base.

Frames. Redundancy can occur in at least four different ways, redundant frames, redundant slots within a frame, redundant content within a frame and redundant connections with other frames. There are at least two approaches that could find use: preventing and detecting redundancies.

A preventive approach would allow a single use of a frame name. The system would not allow further use of any frame name. Within a frame, the same approach could be used to eliminate redundant frame slot names. In order to prevent

redundant connections, the system could allow at most one link hierarchically above and below the frame.

Another approach is to build a system that can detect potential redundancies. The existence of frame redundancies could be determined by comparing the content of all frames from the same hierarchical levels (e.g., as in the figures "group frames"), thus, limiting the number of frames that must be compared. Similarly, within a given frame, redundant slots and contents can be determined by comparing names or contents of slots. Finally, redundancy in links with other frames can be accomplished by examining uniqueness of links to and from frames.

Semantic Nets. Redundancy in semantic nets can occur in at least two ways, redundancy of arcs and redundancy of nodes. As a default, the designer of semantic net systems or shells could set a default of one arc with the same name per node. To the extent that those relationships can be specified, they can be used to assist in the verification process by finding those arcs that do not meet that conditions of the structure placed on them.

For example, in figure 4, depending on the context, it is unlikely that "chair" or any other object will occur in more than one node. Similarly, in many instances the user may wish to limit the use of a particular arc label from a given node to a single (or two or ...) occurrences. For example, a single "isa" may be allowed per node.

4.3 Completeness

A system may violate completeness if there is missing knowledge. The issue then becomes one of determining how to identify missing knowledge. In some situations, the nature of the knowledge representation provides a basis to assert incompleteness. For example, a rule of the form "if a then ----" is incomplete, since there is no conclusion.

Frames. In the case of frames structural incompleteness can occur if there is a missing frame, or a missing slot within the frame or if there is a missing link between frames.

In order to determine if there are missing frames or slots, the list of frame names, slot names and contents are elicited by the system as suggested above, can indicate a lack of completeness. First, if there is an unused frame name, slot name, or content, then that may indicate that the developer "forgot" or "neglected" to include that frame. Second, if there is a frame name, slot name or content that is used, but is not on the list of feasible names then that also can indicate a lack of completeness. Incomplete, since the

use of a name not on a preestablished list is likely to indicate that the lists are incomplete to begin with.

A missing link between frames could be determined using a number of different approaches. If there is no path to a frame and the frame is not a "root" frame, then the knowledge base is not complete. For example, in figure 2 if there was no arc from the room frame to the wall frame then that would be an indication that the knowledge representation was incomplete.

In a similar way, statements can be made about other types of frames such as terminal frames and intermediate frames. If there is no path from the frame and the frame is not a "terminal" frame then the knowledge base is not complete. If the frame is an "intermediate" frame then there should be a path to another frame and from another frame.

Parallel structure also can be used to establish completeness. If the developer knows that there is parallel structure between sets of frames then a comparison between those sets of frames and connections will yield a determination as to completeness.

Semantic Networks. In the case of semantic networks structural incompleteness can occur if there is a node that is not connected by any arc and if there is an arc that leads to no node. For example, in figure 4, if the "isa" arc leads from chair and there is no "furniture" node, then the net is in error. In either case, those conditions indicate either incompleteness or incorrectness.

Further, similar to frames, "root" nodes, "intermediate" nodes and "terminal" nodes establish expectations. If a node is a root node then it should have no incoming arcs and it should only have outgoing arcs. For example, the root node in figure 2, "room frame," has only arcs to other nodes. If a node is an intermediate node then it should have both incoming and outgoing arcs. Similarly, if a node is a terminal node, it should only have incoming nodes and no outgoing nodes.

In addition, there is another approach to assessing completeness. If a list of eligible arcs and nodes is established then at least two cases can indicate a lack of completeness. First, if there is an unused arc or node name then that can indicate that the designer has forgotten to include the arc or node. Second, if there is a name that is used that is not on the list of feasible names, then that also can indicate a lack of completeness, similar to the case of frames, as discussed above.

4.4 Correct Structure

Much of the analysis of correctness of knowledge representation is domain based. However, assuming domain independence, it is still possible to assess the correctness of the structure of the knowledge representation, by drawing on certain graph theory concepts, such as whether the net is a tree or an acyclic graph.

Frames. Typically, relationships between frames employ either a tree structure or acyclic network structure, depending on how the systems fill their slots. Thus, in the first case, verification would consist of determining if the set of relationships between frames define a tree (easily ascertained using network theory). In the second case, verification would consist of determining if the network contains any cycles. This last approach is similar to those used with rule-based systems to elicit cycles, if they exist. Some frame-based packages allow cycles in the frame networks by only testing if the frame has one node hierarchically above and below. Such a test is only a myopic test of whether a graph is acyclic.

There also is a structure within each frame and many such frames employ the same structure. As a result, another important verification approach is to check the structure within frames that should be structured the same: do each have the same named slots, do each have the same number of slots, etc. This approach uses information that is solicited from the user in order to assist in the verification effort. This approach is part of the knowledge acquisition effort, since it can force the user to find symmetry and structure, in the knowledge, to the extent feasible.

Finally, the structure can be used to ascertain when a frame is inappropriately connected to another frame. First, it would be a very unusual situation to have a frame connected to itself. Second, an important verification characteristic of a frame is to collect the planned number of frames that frame is connected to, hierarchically from above and below. Thus, if more than that number of frames are connected in actuality, then there is an error. Again, this approach would try to ensure that the user examined the nature of the knowledge for symmetry.

Semantic Networks. As with frames, semantic networks are likely to employ either a tree approach or an acyclic network approach. Thus, similar verification procedures to frames would be used. Further, it would be unlikely that a node would be connected to itself by an arc. Any such occurrence likely would indicate that there is an error in the knowledge representation.

5. SUMMARY

This paper has argued that although there have been a number of efforts to establish verification processes for rule-based knowledge bases, there has been limited investigation of frame and semantic network knowledge bases. As a result, when building a frame or semantic network-based system or shell, the verification process is not clear from the literature. The thrust of this paper is to mitigate that gap. Thus, this paper provided a number of different approaches to verification of frames and semantic networks. Those approaches were designed to examine the knowledge base for consistency, redundancy, completeness and structure.

These include that network representations of frame-based knowledge bases be employed for a variety of tests, that frame-based systems force the user to established to the system information about parallelism and relationships between frames (e.g., number of frames with inputs to and outputs from, that lists of frame names and slots be used to mitigate errors in such knowledge bases and that frames be established as either root frames, terminal frames or intermediate frames. Similar approaches were developed for semantic networks.

Although this paper has focused on frames and semantic networks, this same approach can be extended to other forms of knowledge representation where the same effort is made to exploit the structure and content of the knowledge representation for verification purposes.

The verification process is critical to knowledge acquisition. At one level, the verification process is necessary to ensure that the knowledge that is represented is correct, complete and consistent. Otherwise when the system is compared to the user there could be errors inhibiting the performance of the system. In addition, this paper has argued that the verification process is an integral part of the knowledge acquisition process. Verification tests were identified that forced the user to examine the knowledge for structural characteristics and symmetry. Such analysis of existing knowledge could lead to additional knowledge.

REFERENCES

Adrion, W., Branstad, M., and Cherniavsky, J., (1982) "Validation, Verification, and Testing of Computer Software," ACM Computing Surveys, Vol. 14, No. 2, pp. 159-192.

Boose, J., (1989) "A Survey of Knowledge Acquisition Techniques and Tools," Knowledge Acquisition, Volume i, pp. 3-37.

Boose, J. and Bradshaw, J., (1987) "Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Knowledge-based Systems," International Journal of Man-Machine Studies, Vol. 26, No. 1, pp. 3-28.

Davis, R., (1976) Use of Meta Knowledge in the Construction and Maintenance of Large Knowledge Bases, Ph.D. Dissertation, Stanford University,.

Gaines, B., "An Overview of Knowledge Acquisition and Transfer," International Journal of Man-Machine Studies, Vol. 26, No. 4, pp. 453-472.

Minsky, M., (1975) "A Framework for Representing Knowledge," in The Psychology of Computer Vision, P. Winston (ed.), McGraw-Hill, New York.

Nazareth, D., (1989) "Issues in the Verification of Knowledge in Rule-based Systems," International Journal of Man-Machine Studies, Vol. 30, pp. 255-271.

Nguyen, T., Perkins, W., Laffey, T., and Pecora, D., (1985) "Checking Expert System Knowledge Bases for Consistency and Completeness," Proceedings of the International Joint Conference For Artificial Intelligence, pp. 375-378.

Nguyen, T., Perkins, W., Laffey, T., and Pecora, D., (1987) "Knowledge-based Verification," AI Magazine, Vol. 8, No. 2.

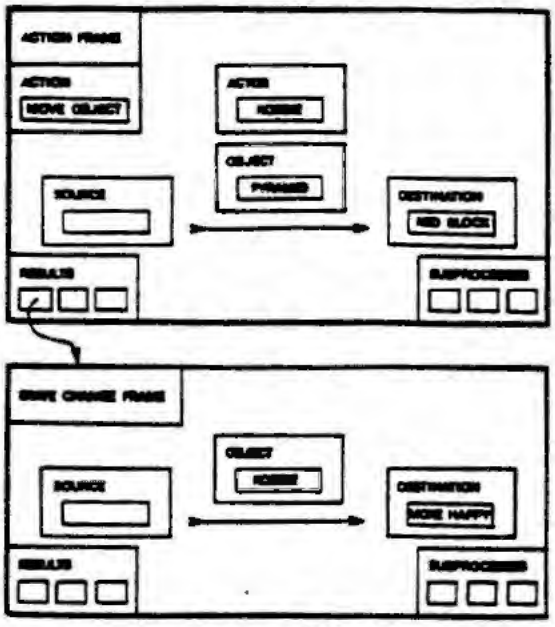
O'Leary, D., (1990) "Soliciting Weights or Probabilities from Experts for Rule-based Expert Systems," International Journal of Man-Machine Studies.

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

Willingham, J. and Ribar, G., (1988) "Development of an Expert Audit System for Loan Loss Evaluation," Auditor Productivity in the Year 2000, Arthur Young.

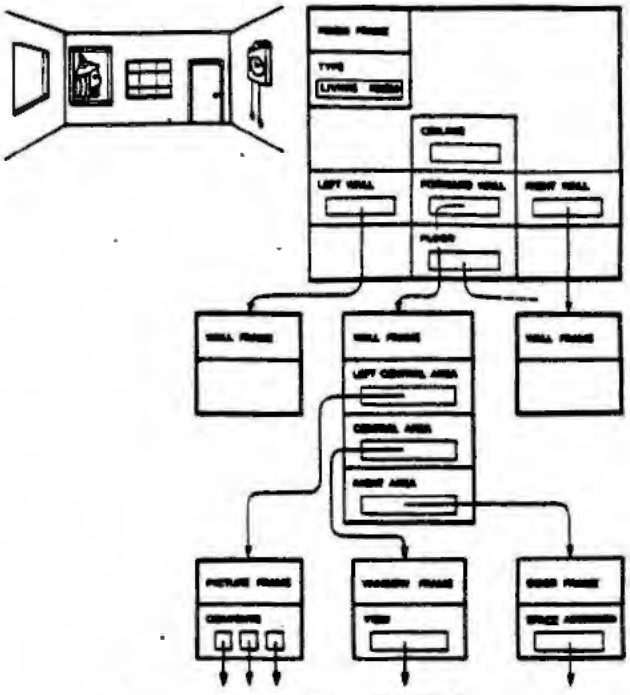
Winston, P., (1979) Artificial Intelligence, Addison-Wesley, Reading, Massachusetts.

Figure 1: Sample Frame



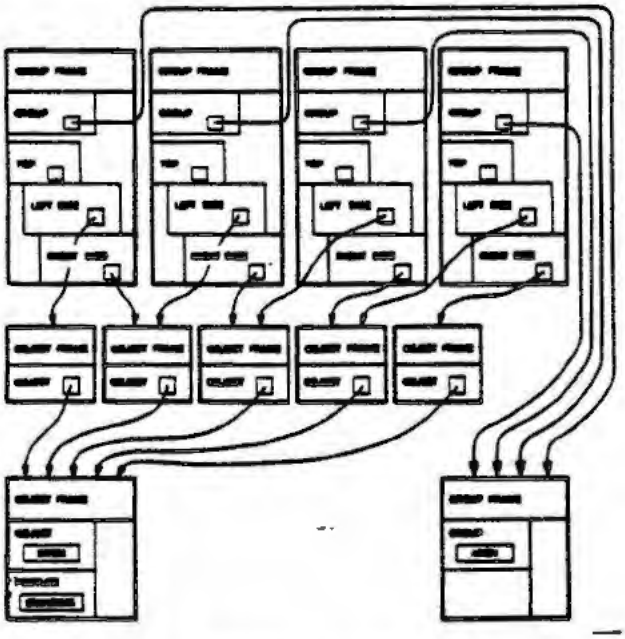
Source: Winston [1979]

Figure 2: Tree Structure of Frames



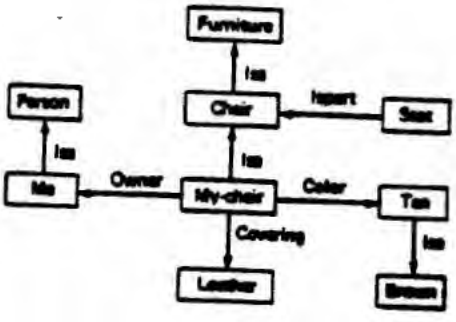
Source: Winston [1979]

Figure 3: Network Structure of Frames



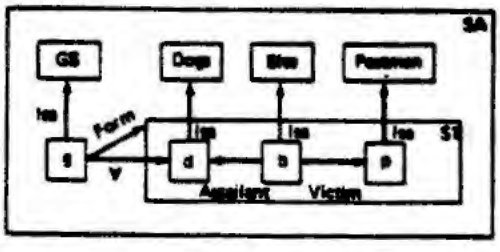
Source: Winston [1979]

Figure 4: Sample Semantic Net



Source: Rich [1983]

Figure 5: Parallel Structure in a Semantic Net



Source: Rich [1983]

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

THE METHOD OF DIRECT KNOWLEDGE ACQUISITION FROM HUMAN EXPERTS.

G.S.Osipov
Laboratory of Knowledge Representation Problems.
Program Systems Institute of the USSR Academy of Sciences ,152140,Pereslavle-Zalessky, USSR.

Abstract

The paper discusses the method of Automated Knowledge Acquisition for structural, assotiative and causal knowledge, which is based on interaction between repertory grid, laddering and some new strategies which is describing in the paper.

Introduction

The system ETS [Boose,1985] is well-known. The method of repertory grid for knowledge acquisition from human experts was used first in this system.

It was used repertory grid and laddering strategies for eliciting traits of concepts and particular concepts in [Diederich,1987].

It is obvious that for solving of some problems it is necessary to elicit relation between concepts.

This paper discusses the method of Automated Knowledge Acquisition for structural, associative and causal knowledge, which is based on interaction between repertory grid, laddering and some new strategies which we will describe later. These new strategies we will called similarity eliciting and similarity confirmation strategies.

In our approach the main elements of human knowledge are: the statements of human experts; names of concepts; attributes of concepts; range of values of attributes and types of relations between concepts.

The work of similarity eliciting and similarity confirmation strategies is based on the eliciting of the properties of statements of experts and on the comparison of these properties with formal properties of some relations. Therefore, before we go to describe the strategies, we must introduce briefly the method of knowledge representation.

The approach, which we describe in this paper was used for implementing the SIMER expertise elication/transfer system.

2. Which types of the statements of human experts suggested method of knowledge representation is based on?

The analysis of literature and our personal experience of communication with experts allowed to select the following types of statements (Table 1).

Table 1

Some types of statements.

Φ_1	d_1 is a characteristic of d_2	Φ_9	d_1 may be accompanied by d_2
Φ_2	d_1 is a manifestation of d_2	Φ_{10}	d_1 can increase possiblity of d_2
Φ_3	d_1 is frequently accompanied by d_2	Φ_{11}	d_1 excludes d_2
		Φ_{12}	d_1 procreate d_2
Φ_4	d_1 increases the possibility of d_2	Φ_{13}	if d_1 is observed d_2 appear
		Φ_{14}	d_1 can procreate d_2
Φ_5	d_1 can occur with d_2	Φ_{15}	d_1 may develop into d_2
Φ_6	d_1 is usually accompanied by d_2	Φ_{16}	d_2 starts with d_1
		Φ_{17}	d_2 may start with d_1
Φ_7	d_1 normally attended by d_2	Φ_{18}	d_2 develops if d_1 is observed
Φ_8	d_1 usually attended by d_2	Φ_{19}	d_2 can develop if d_1 is observed

(here d_1 and d_2 are meta-level variables for the names of concepts).

It is possible to show that the good right interpretation of these and some other types of statements give the non-homogeneous semantic networks [Osipov,1987].

3.Non-homogeneous semantic networks and structural, associative and causal relations

The Non-homogeneous semantic networks N are directed labeled graph whose each vertice is a set of ordered pairs and each arc belongs to some relation and marked by distinguished label. Each arc generally has different algebraic properties, semantics, and hence interpretation.

Formally, $N = \langle D, \tau, \Delta, F, R \rangle$

where

D is the collection of sets $D_1, \ldots, D_n$,

τ is the set of types: $\tau=\{\bar{k}_1, \ldots, \bar{k}_e\}$.

Hear $\bar{k}_1, \bar{k}_2, \ldots$ are ordered subsets of indices contained in the set $\{1,2, \ldots, n\}$.

Let $D_{\bar{k}_i} = D_{n_1} \times \ldots \times D_{n_{k_i}}$ $(n_{k_i} \leq n)$; for each $\bar{k}_i$ $\Delta_{\bar{k}_i}$ is same set of subsets of $D_{\bar{k}_i}$: $\Delta_{\bar{k}_i} = \{d_{\bar{k}_{i_1}}, d_{k_{i_2}}, \ldots, d_{k_{i_j}}\}$. Each element d from $\Delta_{\bar{k}_i}$ (i.e. subset of $D_{\bar{k}_i}$) we will call the event (or formal concept) of type $\bar{k}_i$. Union of all such sets for all $\bar{k}_i$ let us call it Δ: $\Delta = \cup \Delta_{\bar{k}_i}$.

F is the collection of functions: $f_1 \langle \bar{k}_1, p_1 \rangle, \ldots, f_2 \langle \bar{k}_2, p_2 \rangle, \ldots f_t \langle \bar{k}_t, p_t \rangle, \ldots$ where f_i is a function of type $\langle \bar{k}_i, p_i \rangle$ which operates from $D_{n_1} \times \ldots \times D_{n_{k_i}}$ into D_{p_i}, i.e. $f\langle \bar{k}_i, p_i \rangle(d) = m$, with $d \subseteq D_{n_1} \times \ldots \times D_{n_{\bar{k}_i}}$ (d is associated with type $\bar{k}_i$) and $m \in D_{p_i}$; R is the collection of relations on set of events (concepts) Δ.

We will definite the relations from R more detailed; but first in order we will introduce following marks: if d is an event ($d \subseteq D_{\bar{k}_i}$) then the projection of d into set D_j will be denoted by A_j^d and will be called range of values of attribute j of event d. (Note here, that every attribute j of event d really is necessary represent by pair $\langle D_j, A_j^d \rangle$. This information about attributes we can obtained through knowledge acquisition process; we elicit by this process not event $d \subseteq D_1 \times D_2 \times \ldots \times D_k$ but event-prototype $\beta^d = \{\langle D_1, A_1^d \rangle, \langle D_2, A_2^d \rangle, \ldots, \langle D_k, A_k^d \rangle\}$ where $d \subseteq \beta^d$, and A_i^d is the projection of d into D_i. But here we will ignore this fact for simplicity).

Let in following definitions any event d_1 have the type $\bar{k}_1 = \langle n_1, n_2, \ldots, n_{k_1} \rangle$, and any event d_2 have the type $\bar{k}_2 = \langle m_1, m_2, \ldots, m_{k_2} \rangle$.

Definition 1. If for each $A_{m_j}^{d_2}$ from d_2 there exists $A_{n_i}^{d_1}$ from d_1, which satisfies $m_j = n_i$ and $A_{n_i}^{d_1} \subseteq A_{m_j}^{d_2}$, then the pair $(d_1, d_2) \in R_1$.

Definition 2. If there exists $A_{n_i}^{d_1}$ from d_1 and $A_{m_j}^{d_2}$ from d_2 such that $n_i = m_j$ and $A_{n_i}^{d_1} \cap A_{m_j}^{d_2} \neq 0$ and $A_{n_i}^{d_1} \not\subset A_{m_j}^{d_2}$ and $A_{m_j}^{d_2} \not\subset A_{n_i}^{d_1}$, then $(d_1, d_2) \in R_2$.

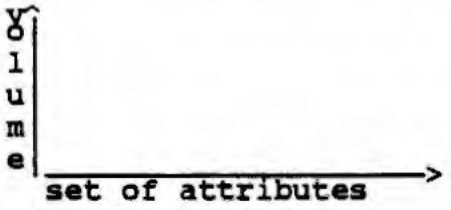
Definition 3. If there exists $A_{n_i}^{d_1}$ from d_1 and $A_{m_j}^{d_2}$ from d_2 such that $n_i = n_j$ and $A_{n_i}^{d_1} \cap A_{m_j}^{d_2} = \emptyset$, than $(d_1, d_2) \in R_3$.

Definition 4. If for each $A_{n_i}^{d_1}$ from d_1 there exists $A_{m_j}^{d_2}$ from d_2 such that $n_i = m_j$ and $A_{n_i}^{d_1} \subset A_{m_j}^{d_2}$, then $(d_1, d_2) \in R_4$.

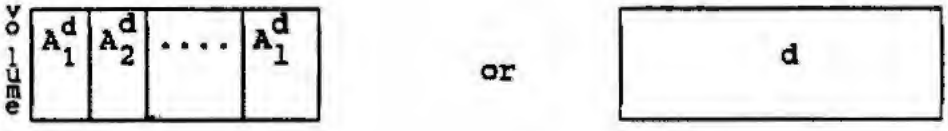
Definition 5. If for each $A_{m_j}^{d_2}$ from d_2 there exists $A_{n_i}^{d_1}$ from d_1 such that $n_i = m_j$ and $A_{n_i}^{d_1} \cap A_{m_j}^{d_2} \neq \emptyset$ and $A_{n_i}^{d_1} \not\subset A_{m_j}^{d_2}$, then $(d_1, d_2) \in R_5$.

Definition 6. If there exists $A_{n_i}^{d_1}$ from d_1 and $A_{m_j}^{d_2}$ from d_2 such that $n_i = n_j$ and $A_{m_j}^{d_2} \subset A_{n_i}^{d_1}$, then $(d_1, d_2) \in R_6$.

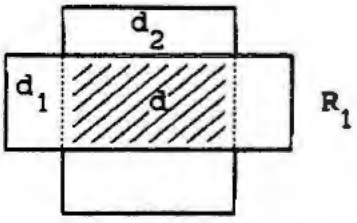
We will illustrate Definition 1-6 by using the diagrams; first of all we will introduce the special "twodimensional" sets, in which the first dimension is the set of attributes and the second dimension is the volume of correspondence attribute, i.e.

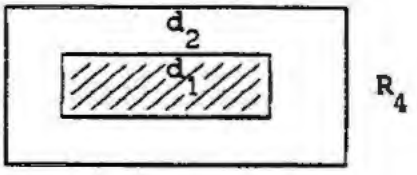


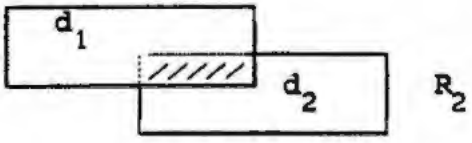
then the each event d is illustrated in the following way:

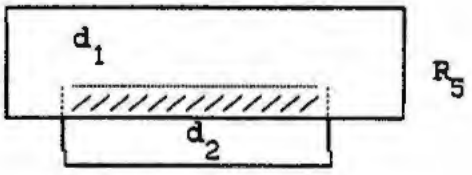


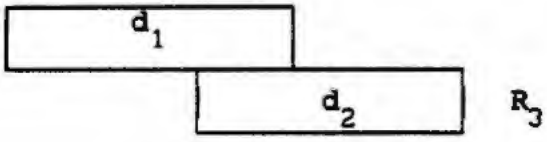
In this case the relations R_1-R_6 may be illustrated:

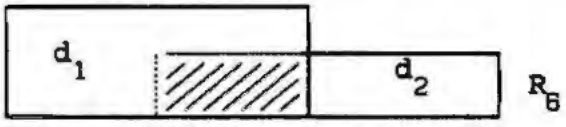












The diagramms R_1-R_6 can be interpreted in the following way:

Lemma 1. $(d_1, d_2) \in R_1$ if for each event-example δ from d_1 there exists event-example γ from d_2 such that $\forall A^{d_2}_{m_j} \exists A^{d_1}_{n_i}$, that $n_i = m_j$ and $a^{d_1}_{n_1} = a^{d_2}_{m_j}$.

Here, the event-example δ from event d_1 of type $\bar{k}$ is an ordered set (suite) $\langle a^{d_1}_1, a^{d_1}_2, \ldots, a^{d_1}_k \rangle$, i.e. $\delta \in d$ and $a^{d_1}_j \in A^{d_1}_j$; analogical the event-example γ from event d_2 is an ordered set $\langle a^{d_2}_1, a^{d_2}_2, \ldots a^{d_2}_n \rangle$, and $a^{d_2}_j \in A^{d_2}_j$.

Lemma 2. $(d_1, d_2) \in R_2$ if $\exists\delta\exists\gamma$ such that $\exists A^{d_1}_{n_i}$ $\exists A^{d_2}_{m_j}$, that $n_i = m_j$ and $a^{d_1}_{n_1} = a^{d_2}_{m_j}$.

Lemma 3. $(d_1, d_2) \in R_3$ if $\forall\delta\exists\gamma$ such that $\exists A^{d_1}_{n_i}$ $\exists A^{d_2}_{m_j}$, that $n_i = m_j$ and $a^{d_1}_{n_i} \neq a^{d_2}_{m_j}$.

Lemma 4. $(d_1,d_2)\in R_4$ if $\forall\delta\exists\gamma$ such that $\forall A^{d}_{n}$ $\exists A^{d}_{m}$, that $n_i=m_j$ and $a^{d_1}_{n_1}=a^{d_2}_{m_j}$.

Lemma 5. $(d_1,d_2)\in R_5$ if $\exists\delta\exists\gamma$ such that $\forall A_m$ $\exists A_n$, that $n_i=m_j$ and $a^{d_1}_{n_1}=a^{d_2}_{m_j}$.

Lemma 6. $(d_1,d_2)\in R_6$ if $\forall\gamma\exists\delta$ such that $\exists A_n$ $\exists A_m$, that $n_i=m_j$ and $a^{d_1}_{n_1}=a^{d_2}_{m_j}$.

It is possible to demonstrate, that the relations R_1 - R_6 have different algebraic properties. Namely if we denote the algebraic properties: transitivity - t,non-transitivity - n, reflexivity - r, irreflexivity - i, non-reflexivity - x, symmetry - s, antysymmetry - a, non-symmetry - y then:
R_1 is t,r,a; R_2 is n,r,s; R_3 is n,i,s;
R_4 is t,i,a; R_5 is n,r,y; R_6 is n,i,y.

The relations R_1 to R_6 and opposite relations we will call the structural relations.

If we introduce the relations R_7 to R_{12}, which have the same properties as R_1 to R_6, but doesn't satisfy to definitions 1 to 6, then these relations may be called associative relations.

Let us introduce now the set of causal relations.

Let in following definitions any events d_1 and d_2 have as before the types $\bar{k}_1$ and $\bar{k}_2$ respectively.

Definition 7. If for every event-example δ_1 of event d_1 there exists event-example δ_2 of event d_2 so that for every attribute A_i from d_2 there exists function f from F and segment $\delta'\subseteq\delta_1$ that $f(\delta')=a_i$ (where a_i is a value of attribute A_i from δ_2), then $(d_1, d_2)\in R_{13}$.

It is possible to give the Definitions by other causal Relations R_{14} - R_{17} analogously.

For example we cite the Definition for relation R_{14}.

Definition 8. If there exists event-example δ_1 of event d_1 and event-example δ_2 of event d_2 so that for some attributes A_j from d_2 there exists function f from F and segment $\delta'\subseteq\delta_1$ that $f(\delta')=a_j (a_j\in A_i)$, then $(d_1, d_2)\in R_{14}$.

If the members of collection F are treated as time functions, i.e. producing their output values in the time moment that follows moment of time of their argument, then the Relations R_{13} to R_{17} on the set of events are generated by the following process: in the moment of time t_6 one or more attributes assume

II. It is applied suitable rules from rules 1.1-1.3 to the quantifier prefix of lemma.

III. It is applied suitable rules from 2.1-2.3 to the quantifier prefix of lemma.

IV. It is applied rules from 3.1-3.2 to result of applying of rules of 1.1-1.2.

V. It is applied rules from 3.3-3.4 to the result of rules 2.1-2.2.

VI. The following construction is formed: "<name of the first event><wright part of suitable rule from 1.1-1.3><wright part of suitable rule from 2.1-2.3><name of the second event> and <name of the second event><application of the rule from 3.1-3.2 to the rule from 1.1-1.3, applied on step II><application of the rule from 3.3-3.4 to the rule from 2.1-2.3, applied on step III><name of the first event>.

We will name the changing of the wright part of applied rules from 1.1-1.3 and from 2.1-2.3 in accordance with rules 3.1-3.4 as application of rules.

If $a_{n_1}^{d_1} \neq a_{m_j}^{d_2}$ so the suitable associative relations is translated to the statement of following type <name of the first event> excludes <name of the second event> and <name of the second event> excludes <name of the first event>>>. For example we shall apply the rule of translation and substitution to relation R_2. For this purpose in lemma 2 we change condition $\exists\delta\exists\gamma$ according to rule 1.3 to "can" and condition $\exists A_{n_1}^{d_1}\exists A_{m_j}^{d_2}$ according to rule 2.3 to "increase of the possibility".

Further according to substitution rules will recieve:"d_1 can increase the possibility of d_2" and " d_2 can increase the possibility of d_1".

In this way R_2 is translated to the pair statements Φ_{10} and Φ_{10}^{*} (through Φ_i^{*} we will denote Φ_i with the substitution of d_1 for d_2 and d_2 for d_1).

If $a_{n_1}^{d_1} \neq a_{m_j}^{d_2}$, any concurrency relation reduces to the sentence: "d_1 excludes d_2" and "d_2 excludes d_1".

Thus the Theorem holds:

R_1 corresponds to Φ_{12}^{*} and Φ_{10}^{*};

R_2 corresponds to Φ_{10} and Φ_{10}^{*};

R_3 corresponds to Φ_{11} and Φ_{11}^{*};

R_4 corresponds to Φ_4 and Φ_{14}^{*};

R_5 corresponds to Φ_{14} and Φ_{10}^{*};

specific values $<a_1, a_2, \ldots, a_k>$; appropriate function from F are applied to all events included these attributes and generate values of new attributes in the moment of time t+1.

Relation R_{13}-R_{17} are established a correspondence for each event associated with new attributes.

We will suggest that the members of collection F not contain identify mapping and inverse functions; then it is possible to show that Relations R_{13}, R_{14} (and R_{15}-R_{17}) haven't properties of reflexivity and irreflexivity.

Namely:

R_{13} is t,x,y;

R_{14} is n,.x,y.

Using of Lemmas 1-6 give us the possibility of researching of the correspondence between Relations R_1-R_{17} and the Statements from Table 1. Namely, each Relation corresponds to pair of Statements.

For this we define special rules to relate the notation of Lemmas 1-6 to quantifiers as inserting the resultant expressions into a structure we describe below.

The rules are called the translation rules and the substitution rules, respectively.

Translation Rules

Rule I

1.1. $\forall\delta \;\; \exists\gamma \Rightarrow$ 'always'

1.2. $\forall\gamma \;\; \exists\delta \Rightarrow$ 'can'

1.3. $\exists\delta \;\; \exists\gamma \Rightarrow$ 'can'

Rule II

2.1. $\forall A_{m_j}^{d_2} \;\; \exists \; A_{n_i}^{d_1} \Rightarrow$ 'be procreate of '

2.2. $\forall A_{n_i}^{d_1} \;\; \exists A_{m_j}^{d_2} \Rightarrow$ 'increase of the possibility '

2.3. $\exists A_{n_i}^{d_1} \;\; \exists A_{m_j}^{d_2} \Rightarrow$ 'increase of the possibility '

Rule III

3.1. Wright part of 1.1 exchange to wright part of 1.2
3.2. Wright part of 1.2 exchange to wright part of 1.1
3.3. Wright part of 2.1 exchange to wright part of 2.2
3.4. Wright part of 2.2 exchange to wright part of 2.1

Substitution Rule (case $a_{n_i}^{d_1} = a_{m_i}^{d_2}$)

Which from associative relations is translated to one type of statements by using following procedure:

1.It is chosen one of lemm 1-6.

R_6 corresponds to Φ_{10} and Φ_4^* ;
R_{13} corresponds to Φ_{12} and Φ_{17};
R_{14} corresponds to Φ_{15} and Φ_{17}.

This correspondence is the mapping from R into $(\Phi \cup \Phi^*) \times (\Phi \cup \Phi^*)$, where R is set of relations; Φ is set of types of statements from Table 1. (through Φ we will denote Φ with the substitution of d_1 for d_2 and d_2 for d_1) .

The next problem is the search of the mapping from $(\Phi \cup \Phi^*) \times (\Phi \cup \Phi^*)$ into R.

This problem is the Task of direct knowledge acquisition and discusses in following part.

4. Knowledge Acquisition

The Knowledge Acquisition Process includes the next phases:
- Structural Knowledge eliciting/transfer;
- Associated Knowledge eliciting/transfer;
- Causal Knowledge eliciting/transfer.

Laddering, repertory grid, similarity eliciting and similarity confirmation strategies are used for eliciting the structural knowledge.

The repertory grid and laddering strategies was described in [1] and [2] and therefore we will discuss briefly the similarity eliciting and similarity confirmation strategies.

Similarity eliciting strategy. This strategy is based on the eliciting of some properties of statements of Human experts in the interactive work. These properties are:

-invariance (or not) of truth of the statement with substitution of names of concepts;

-truth (or not) of statement, which is conjunction from two true statements with one common name of concept.

On the basis of these properties and properties of relations from R the hypothesis about belonging the statement to some kind Relations R_k from collection R is formed. For example, suppose the Knowledge Base have the next events:

-phagocytic activity leukocytes; -suppuration;
-herpetic eruption; -painful urincetion;
-pneumonia; -renal colic;
-pyuria; -pyelonephrities.

(Let these events is obtained by means of laddering or repertory grid strategies). The system asks the expert to respond "Which events from the list is connected among themselves?" The expert now selects pyuria and pyelonephritis from the menu. The system then asks:

1. Is pyuria a manifestation of pyelonephritis?
2. May pyuria be accompanied by pyelonephritis?
3. Is pyuria a negative manifestation of pyelonephritis?
4. Does pyuria exclude of pyelonephritis?

The expert selects answer 1 from menu now.

Then the system asks:

1.Is pyelonephritis a manifestation of Pyuria?

2.Does pyuria increase possibility of pyelonephrity?

3.Can pyuria increase possibility of pyelonephrity?

If the expert selects answer 3 from menu then the hypothesis that the pair of events (pyuria, pyelonephritis) belong to Relation R_1 (Table 2) forms in knowledge base. Increment of likelihood of this hypothesis is attained by means of similarity confirmation strategy.

Similarity confirmation strategy. This strategy is combined strategy and based on the interaction of repertory grid and similarity eliciting strategies.

Similarity confirmation strategy makes attempt to confirm the hypothesis which was obtained by similarity eliciting strategy. This attempt is based on the Definitions 1.-8. where the belonging to relations from collection R is determined through the attributes of events. In order that determine the Attributes of events the repertory grid strategy is used.For example for events pyuria and pyelonephritis the system selects the third event which is not connected with the first and the second events. Let this event is herpetic eruption.

The system asks the expert:

What is a trait which characterizes difference between herpetic eruption and pair of events: pyuria and pyelonephrities?

After the Expert's respond the system asks the series of questions: about opposite trait, about the common name for the trait and the trait opposite him, about the range of values of new attribute with common name. Then instead of event herpetic eruption the system chooses other event and is realized the next iteration. After every iteration the system make the attempt to application the Definition 1. (In general - the Definitions 1-8).

In case of success the hypothesis about belonging the pair (pyuria, pyelonephritis) to Relation R_1 is certain assertion; else the work of repertory grid strategy for new traits eliciting is repeated.

If in spite of these efforts the condition of certainty is not realized then the hypothesis is preserving in Knowledge Base but it is marked as incorrect.

This mark restricts the possibility of use of this connection. For example on this connection not arise inheritance of traits.

By work of these strategies the scenario of working determined by the system.

5. Inheritance mechanism

Inheritance of traits usually can be used for default transfer of traits of generally concepts to particular concepts. Usually it used in automatic behavior in deductive inference mechanisms.

We obtain the possibility of using of this mechanism in Knowledge acquisition process. But its application has some peculiarities. The first peculiarity is: in badly structured

domains inheritance is not certain but only likely. It means that inheritance mechanism must be implemented as interactive procedure. The second peculiarity is: interactive implementation inheritance mechanism allows some extension. Namely in case interactive working it is possible hypotheses generation about inheritance of traits along tolerance, similarity types of Relations, i.e. along Relations R_2, R_4, R_5, R_6 and other but not only along R_1 -partial order Relation. For example, if event d_1 have the trait A and the pair $(d_1,d_2) \in R_2$ then forms the hypothesis that the event d_2 has the trait A. If the Expert responsed "Yes" the trait A is ascribed to event d_2.

Conclusion

The result of the work of this strategies is the Knowledge Base on the Non-homogeneous Semantic Networks. We designed the special non-deductive inference mechanism which simulates reasoning of type of argumentation over this knowledge base.

The interaction of the inference mechanism with repertory grid, laddering, similarity eliciting and similarity confirmation strategies allows elicit the special Knowledge for some special tasks too. But their describing falls outside the limits of this paper.

References

1. Boose, John H.(1985) Personal Construct Theory and the transfer of Human Expertise. Advances in Artificial Intelligence, T.O'Shea(Ed.),1985, Elsevir Science Publishers B.V.(North-Holland). ECCAI.

2. Diederich,J., Ruhmann, I. & May, M. (1987) KRITON: a knowledge acquisition tool for expert systems,1987, International Journal of Man-Machine Studies. Special Issue.

3. Osipov G.S., Kurshev E.P. ,Komarov S.I. ,Gucova S.M. (1987) Expert systems tools for badly structured fields. Artificial Intelligence and Information-Control Systems of Robots-87. I.Plander (Ed.),1987, Elsevir Science Publishers B.V.(North-Holland), P.393-397.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Detecting and Correcting Errors in Rule-Based Expert Systems: An Integration of Empirical and Explanation-based Learning

Michael J. Pazzani and Clifford A. Brunk
Department of Information and Computer Science
University of California,
Irvine, CA 92717
714-856-5888
(pazzani@ics.uci.edu)
(brunk@ics.uci.edu)

Abstract

In this paper, we argue that techniques proposed for combining empirical and explanation-based learning methods can also be used to detect errors in rule-based expert systems, to isolate the blame for these errors to a small number of rules and suggest revisions to the rules to eliminate these errors. We demonstrate that FOCL, an extension to Quinlan's FOIL program, can learn in spite of an incorrect domain theory (e.g., a knowledge base of an expert system that contains some erroneous rules). A prototype knowledge acquisition tool, KR-FOCL, has been constructed that can utilize a trace of FOCL to suggest revisions to a rule base.

Introduction

Buchanan et al. (1983) provide an overview of the expert system development process. After identifying problem characteristics, designing representations for the available data and the system's hypotheses, a prototype is constructed. Next, a cycle is repeated in which the prototype is expanded to cover more problems, tested and revised as necessary to improve accuracy on known problems. Occasionally during this testing phase, it is necessary to reconsider the choices made for representation or to clarify the problem being addressed.

Knowledge acquisition tools excel at the creation of a prototype. For example, ETS (Boose, 1984) automates the processes of eliciting problem characteristics, identifying important domain relations, and generating initial rules for a prototype. However, once the initial prototype is constructed, the rules must be revised manually. Similarly, KNACK (Klinker et al., 1989) and SALT (Marcus, McDermott & Wang, 1985) assist in interviewing domain experts to elicit the knowledge required to instantiate a rule-based expert system, but do not fully address the revision of the rule base when errors are discovered.

TEIRESIAS (Davis, 1978) supports the expert and knowledge engineer in revising a rule base. It provides tools to explain the inference path taken by an expert system. If an expert believes this line of reasoning to be erroneous, the rules involved can also be modified by use of TEIRESIAS. This knowledge refinement tool can assist the user in modifying a rule by use of general rule models that indicate what type of information a rule should be encoding. However, much of the effort is left to the expert and knowledge engineer to assign blame for an error to a particular rule, to identify what particular revisions are necessary and to evaluate the impact of making these revisions.

We are interested in providing a tool to help automate the process of revising a prototype when an error is encountered. We restrict our attention to backward-chaining rule-based expert systems that perform classification tasks (Clancey, 1984). The tool, KR-FOCL partially automates the task of identifying rules responsible for errors in expert systems. KR-FOCL is based on a machine learning program, FOCL, developed at the University of California, Irvine (Pazzani & Kibler, 1990). FOCL was designed to learn constant-free Horn-Clause concepts by combining explanation-based and empirical learning. As a consequence of this combination of learning methods, FOCL can tolerate incomplete and incorrect domain theories (Mitchell, Keller, & Kedar-Cabelli, 1986; Rajamoney, & DeJong, 1987). FOCL uses the rule base of an expert system as the domain theory required by explanation-based learning and processes a collection of examples. If the rule base correctly classifies every example, then FOCL creates an operational definition of the classifications performed by the expert system by using only explanation-based learning techniques.[1] On the other hand, if the rule base incorrectly classifies some examples, FOCL will use some combination of explanation-based and empirical learning techniques. In this paper, we show that the conditions learned by the empirical techniques and a description of when the empirical techniques are needed provides

1. Note that we are not advocating replacing the rule base of the expert system with the operational definition of the concept created by explanation-based learning. This operational definition could be used to classify new examples. However, since it removes all of the intermediate hypotheses, the operational concept will be of little use in explaining how the expert system solved a particular problem (Swartout, 1981).

information that focuses the assignment of blame for an error to a particular rule and the revision of erroneous rules.

Table 1 contains a trace of an interaction with KR-FOCL. In this example, FOCL was given a rule-base of a small expert system designed to indicate when a student is required to pay back an educational loan. This expert system was constructed by an undergraduate student at the University California, Irvine who was employed as a loan processor. Four errors were intentionally introduced into the rule base. Appendix I contains a listing of the expert system rule base. FOCL is also given a collection of 48 examples. The examples were evenly divided between those students who were required to make a loan payment and those who were not. With the errors introduced into the rule-base, 21 of the 24 positive examples and 11 of the 24 negative examples are processed correctly by the expert system. The revisions to the rule-base will focus on changing the rules responsible for misclassifying these examples.

The expert system uses a LISP representation for rules. The first element of a list is a predicate name. The remaining elements are arguments to the predicates. Variables are preceded by question marks. For example, the term:[2]

```
(AND (filed_for_bankruptcy ?s) (disabled ?s))
```

can be read as "The student has filed for bankruptcy and the student is disabled."

Note that KR-FOCL is intended to be used by someone familiar with the domain. This person must approve the changes suggested and help with assignment of blame after KR-FOCL has localized blame to a small number of rules.

In the remainder of this paper, we first describe the learning system, FOCL. Next, we discuss the class of errors in a rule base that FOCL can tolerate. Finally, we report on how KR-FOCL can be used to revise a knowledge-base.

Background: FOIL and FOCL

In order to learn the complex concepts encoded in the knowledge-base of an expert system, it is necessary to have a learning program capable of acquiring relational concepts. Recently, Quinlan (1990) has introduced a learning system called FOIL that is capable of learning Horn Clause concepts such as those used by backward-chaining rule interpreters. FOIL is a purely inductive system. We have produced an extension to FOIL, called FOCL, that combines the inductive component of FOIL with an explanation-based learning component.

---

2. Note that we have edited FOCL's notation slightly to improve readability. In particular, FOCL uses the position of elements in a rule to indicate the antecedent, the consequent and the presence of conjunction. That is, instead of:

```
(between ?x ?y) IF (AND (less-than ?x ?z)(less-than ?z ?y))
```

FOCL uses:

```
((between ?x ?y ?z) (less-than ?x ?y)(less-than ?y ?z)).
```

In addition, since FOCL uses constant-free literals, so that

```
(less-than ?x 5)
```

must be represented as:

```
(AND (less-than ?x ?y)(five ?y)).
```

Table 1. A trace of KR-SaranWrap

```
1a. Should condition (not_enrolled_for_six_or_more_months ?s) be added to
clause 0 of enrolled_in_more_than_five_units:
(AND (enrolled ?s ?school ?units) (school ?school) (> ?units 5))
> no

1b. Should conditions (not_enrolled_for_six_or_more_months ?s) be added to
clause 0 of continuously_enrolled:
(enrolled_in_more_than_five_units ?s)
> yes

2. Should clause 0 of eligible_for_disability_deferment:
(AND (filled_for_bankruptcy ?s) (disabled ?s))
 be replaced with (disabled ?s)
> yes

3a. Should condition (unemployed ?v1) be added as a new clause for
no_payment_due
> no

3b. should condition (unemployed ?v1) be added as a new clause for
eligible_for_disability_deferment
> no

3c. Should condition (unemployed ?v1) be added as a new clause for
eligible_for_financial_deferment
> yes

4. Clause 4 of eligible_for_deferment:
(eligible_for_disability_deferment ?s)
 was unused.  note: (eligible_for_disability_deferment ?s) has been revised
 should it be deleted?
> no

5. Clause 0 of eligible_for_financial_deferment:
(AND (enrolled ?s ?c ?u) (uci ?c))
 was unused.  Should it be deleted?
> yes
```

Foil

FOIL (Quinlan, 1990) inductively generates constant-free Horn-clause theories in a manner similar to that used by ID3 (Quinlan, 1986) to generate decision trees with attribute-value tests. In particular, FOIL uses a divide-and-conquer approach guided by a heuristic based on information theory. In order to review Quinlan's approach, we need to introduce some terminology. A Horn-clause definition for a concept $(P_0 \ldots ?V_{0,1} \ ?V_{0,n_0})$ consists of a disjunctive set of clauses. Each clause consists of a head and a conjunction of literals:

$(P_0\ ?V_{0,1} \ldots ?V_{0,n_0})$ if (AND $(P_1\ ?V_{1,1} \ldots ?V_{1,n_1}) \ldots (P_m\ ?V_{m,1} \ldots ?V_{m,n_m})$).

We will call $(P_m\ ?V_{m,1} \ldots ?V_{m,n_m})$ a variabilization of a predicate P_m. Since FOIL is currently under development, we present here only its essential characteristics. FOIL is given a collection of labeled examples, a set of known predicates, and an unknown predicate P_0 of known arity that FOIL is to learn. The task is to determine a Horn-clause definition of P_0 in terms of the given predicates as well as (in a limited manner) the predicate P_0.

In our description of FOIL, Pos is the set of positive instances and Neg is the set of negative instances. Similar to AQ (Michalski, 1980), FOIL has two main phases: adding clauses to the theory until every positive instance is covered and forming clauses that do not contain any negative instances. Positive instances are covered as follows:

```
Until Pos is empty
      Construct a clause that covers some positive instances
            and avoids all negative instances.
      Add  clause to the theory.
      Remove those elements of Pos that are covered by the new clause.
```

Constructing a clause that misses all negative instances can also be described simply. Note that the clause $(P_0\ ?V_{0,1} \ldots ?V_{0,n_0})$ if true covers all positive instances. To avoid negative examples, this clause is specialized in the following manner:

```
Let   (P0 ?V0,1 ... ?V0,n0) if true be the initial clause
Let Pos be the positive examples not satisfied by the current definition
Let Neg be the negative examples
Until Neg is empty
      Choose the predicate variabilization with the maximum information
gain.
      If the maximum information gain ≤ 0, then exit with failure
      Conjoin the  predicate variabilization the body of the clause.
      Let Pos be all extensions of Pos that are satisfied by the term.
      Let Neg be all extensions of Neg that are satisfied by the term.
```

At this level of abstraction, FOIL is quite simple. It uses hill-climbing to add the literal with the maximum information gain to a clause. For each variabilization of each predicate, FOIL measures the information gain, which is defined differently from ID3. Without going into the exact computation, (see Quinlan, 1990) we note that, in effect, the information metric determines the number of positive and negative examples satisfied and those predicate variabilizations that correctly classify more examples have the higher information gain.

FOCL

Our work on FOCL is motivated by three factors:

1. FOIL is a purely inductive learning system and cannot make use of any available domain knowledge. When there is domain knowledge available, analytic learning

systems can learn from fewer examples than inductive learning systems because the domain knowledge can be used to constrain the hypothesis space.

2. The hill-climbing control structure of FOIL can prevent it from learning some concepts. For example, both `(less-than ?a ?b)` and `(less-than ?b ?c)` may have negative information gain, while the conjunction of these two terms may have positive information gain. As a consequence, FOIL is not able to learn the conjunction of these two literals.

3. The size of the hypothesis space searched by FOIL can be quite large (see Pazzani & Kibler, 1990 for an analysis of the complexity of FOIL). FOIL computes the information gain of all orderings of all the bound variables and new variables for all known predicates. By using domain knowledge, it may be possible to prune the number of predicate variabilizations considered.

FOCL, like FOIL, uses an information-based metric to evaluate when to add a literal to a clause under construction. However, unlike FOIL literals may be proposed either by an inductive component or by an explanation-based component. The result of this integration is that the system can make use of domain knowledge when available to constrain the search for a concept definition. However, when the domain knowledge is incomplete or incorrect, the literals proposed by the explanation-based component will have less information gain than the literals proposed by the inductive component. As a consequence, FOCL utilizes incomplete and incorrect domain theories and can make up for these deficiencies by relying on an inductive component to complete the concept definition.

Like all explanation-based learning systems, FOCL requires a target concept, (i.e., a non-operational definition of the concept to be required), a domain theory theory (i.e., a set of rules, such as the rule base of the expert system, that relates the non-operational definition to operational predicates), a set of operational predicates (i.e., features that the examples are expressed in terms of), and a set of classified training examples.[3] Table 2 displays this information for the student loan example and a positive and a negative training example. Note that the domain theory in Appendix I incorrectly classifies these two examples. S3 is erroneously classified as a negative example (because the rule for disability deferment has been modified by adding an extra condition) and n17 is incorrectly classified as a positive example (because an extra clause has been added that states that students enrolled at UCI are eligible for a financial deferment).

FOCL differs from FOIL only in how FOCL selects literals to add to a clause. FOCL first computes the information gain of each literal in the target concept. In Table 2, the target concept is a single literal. In general, the target concept can be a conjunction of literals of the form:

$$(\text{AND } (P_1\ ?V_{1,1} \ldots ?V_{1,n_1}) \ldots (P_m\ ?V_{m,1} \ldots ?V_{m,n_m})).$$

If any literal has positive information gain, then FOCL operationalizes that literal. Otherwise, FOCL uses the inductive method of FOIL and computes the information gain of all

3. Note that FOIL only requires a set of operational predicates and a set of training examples.

Table 2. Input to SaranWrap

```
Target concept:        (no_money_owed ?s) if (no_payment_due ?s)
Domain theory:         (see Appendix I)
Operational Pred:      ( (enrolled  ?student ?school ?units)
                         (longest_absence_from_school ?student ?months)
                         (enlist ?student ?organization)
                         (unemployed ?student) filled_for_bankruptcy
                         (filled_for_bankruptcy ?student)
                         (disabled ?student)
                         (school ?organization)
                         (armed_forces ?organization)
                         (peace_corps ?organization) )
Positive Examples:     (s1,s2...,s24)
Negative Examples:     (n1,n2...,n24)

      A Positive Example                    A Negative Example
(longest_absence_from_school  s3 3)   (longest_absence_from_school n17 12)
(enrolled s3 OCC 5)                   (enrolled n17 UCI 10)
(disabled s3)
```

variabilizations of all predicates (both operational and nonoperational) and selects the predicate variabilization with the maximum information gain. This strategy insures that FOCL uses as much of the user provided domain theory as possible, but disregards those parts of the domain theory that misclassify large numbers of examples.

The operationalization process in FOCL differs from that of EBL in that it is guided by an information gain metric over a set of both positive and negative examples rather than a single positive example. As in EBL, the operational definition for a predicate may specialize the predicate if the domain theory is disjunctive (i.e., if there are multiple clauses for any non-operational predicate). In EBL, the predicates that are the leaves of the proof tree of the single training example are used as the operational definition. In FOCL, the information gain metric is used to determine how to expand a proof tree, in the following manner:

```
operationalize(Term, Pos, Neg):
let Body be the empty set.
for each Clause in the definition of Term
   compute_gain(Clause, Pos, Neg)
for the Clause with the maximum gain
   for each term T in Clause
        if T is operational
        then add T to Body
        else add operationalize(T, Pos, Neg) to Body
```

A final difference between FOCL and Foil is that the inductive component of FOCL can make use of nonoperational predicates. The inductive component of FOCL computes the

information gain of all possible variabilizations of the nonoperational predicates in the same manner as the operational predicates. If a nonoperational predicate has the maximum information gain, it is operationalized in the same manner that the target concept is operationalized by the explanation-based learning component of FOCL.

An example will help to illustrate how FOCL operates. Later, we will use this same example to show how KR-FOCL can be used to revise an expert system. The trace in Table 3 shows FOCL operating with the incorrect domain theory from Appendix I. In the first step, FOCL tries to operationalize the concept `no_payment_due`. There are two clauses that can be used to prove that no payment is due and FOCL computes the information gain of both and selects the alternative with the highest information gain: `eligible_for_deferment`. There are five alternative ways of proving this and `eligible_for_military_deferment` has the highest gain. Finally, this rule has only one alternative. Since this alternative is operational, it is added as a new literal. The new literal does not cover any negative examples, so it is a new clause. In Steps 2-4, this process is repeated and rules corresponding to `eligible_for_financial_deferment`, `eligible_for_student_deferment`, `eligible_for_peace_corps_deferment` are operationalized.

In Step 5, the predicate `continuously_enrolled` is operationalized. However, the literal formed by operationalizing this predicate, also covers thirteen negative examples (i.e., the expert system would erroneously report that these students should not repay their loans, but they should). At this stage, FOCL tries to induce another literal that excludes these negative examples, and satisfies some of the positive examples that are covered by the operationalized literal for `continuously_enrolled`. The literal, `never_left_school`, has the highest information gain. Because this predicate is not operational (i.e., it is defined by a rule rather than a set of facts that are true of students), this predicate is also operationalized. The resulting literal is conjoined with the operationalization of `continuously_enrolled` and a new clause is formed that does not cover any remaining examples. Note that the first part of this clause was formed by EBL (operationalizing a target concept with positive information gain) and the second part was formed inductively (by searching all variabilization of all operational and all nonoperational predicates).

At this point, the target concept no longer has positive information gain. This occurs because the set of positive examples is reduced by eliminating those examples that are satisfied by each new clause created. Since the target concept no longer will correctly classify the remaining examples, only inductive techniques are utilized. In Step 6, it is induced that disabled persons are not required to pay back loans and in Step 7, it is determined that unemployed persons are not required to make loan payments. At this point, all positive examples are covered by some clause, and no negative examples are covered by any clause, so the process terminates.

In this example, FOCL first operationalized as much of the domain theory as possible. Note that the first few clauses do not require any induction. Later clauses operationalize part of the domain theory, but use induction to add extra literals. This is a sign that the domain theory is close to being correct. Finally, FOCL uses only inductive techniques to learn the final clauses. This behavior is common and it is a consequence of using information gain as a metric to guide a greedy search for a complete set of clauses that cover all positive and no negative examples.

Table 3. A trace of SaranWrap

```
1. >(continuously_enrolled ?v1)                                    gain    1.4
>(eligible_for_deferment ?v1)                                      gain   14.0
>operationalizing (eligible_for_deferment ?v1)
>>(eligible_for_military_deferment ?v1)                           gain    6.0
...
>>(eligible_for_disability_deferment ?v1)                         gain    1.0
>>operationalizing (eligible_for_military_deferment ?v1)
>>>(AND (enlist ?v1 ?v101) (armed_forces ?v101))                  gain    6.0

New Literal: (AND (enlist ?v1 ?v101) (armed_forces ?v101))
New Negatives: {}

New Clause: (AND (enlist ?v1 ?v101) (armed_forces ?v101))

2. New Clause: (filled_for_bankruptcy ?v1)
3. New Clause: (AND (enrolled ?v1 ?v103 ?v102)(school ?v103)(> ?v102 11))
4. New Clause: (AND (enlist ?v1 ?v101) (peace_corps ?v101))
5. >(continuously_enrolled ?v1)                                    gain    1.8
>(eligible_for_deferment ?v1)                                      gain    0.0
>operationalizing (continuously_enrolled ?v1)
>(enrolled_in_more_than_five_units ?v1)                           gain    1.8
>operationalizing (enrolled_in_more_than_five_units ?v1)
(and (enrolled ?v1 ?v103 ?v102) (school ?v103) (> ?v102 5))       gain    1.8
New Literal: (AND (enrolled ?v1 ?v103 ?v102) (school ?v103) (> ?v102 5))
New Negatives: {n23 n22 n20 n18 n17 n16 n15 n14 n13 n12 n11 n10 n9}

>(enrolled ?v1 ?v104 ?v105)                                        gain    0.0
>(enlist ?v1 ?v106)                                               gain    0.0
>(disabled ?v1)                                                   gain    4.5
>(filled_for_bankruptcy ?v1)                                      gain    0.0
...
>(never_left_school ?v1)                                          gain    6.1
>operationalizing (never_left_school ?v1)
>(AND (longest_absence_from_school ?v1 ?v107)(> ?v108 6))         gain    6.1
New Literal: (AND (longest_absence_from_school ?v1 ?v107)(> ?v108 6))
New Negatives: {}
New Clause: (AND (enrolled ?v1 ?v103 ?v102) (school ?v103)(> ?v102 5)
                 (longest_absence_from_school ?v1 ?v107)(> 6 ?v107))

6. >(continuously_enrolled ?v1)                                    gain   -2.7
>(eligible_for_deferment ?v1)                                      gain    0.0
...
>(disabled ?v1)                                                   gain    9.2
>(unemployed ?v1)                                                 gain    4.6
...
New Literal: (disabled ?v1)
New Negatives: {}
New Clause: (disabled ?v1)
7. New Clause: (unemployed ?v1)
```

Errors in rule bases

KR-FOCL is designed to deal with four types of errors in a rule base:

1. A clause has an extra literal (or literals) added. This extra literal makes the rule overly specific so that some positive examples will be classified incorrectly.

2. A clause has a literal (or literals) missing. This missing literal makes the rule overly general so that some negative examples will be classified incorrectly.

3. A rule has an extra clause (or clauses) added. This extra clause makes the rule overly general.

4. A rule has an clause (or clauses) missing. This missing clause makes the rule overly specific.

The rule base displayed in Appendix I has one of each type of these errors. Note that combinations of these errors can account for other erroneous rules. For example, using the wrong literal (e.g., `(> ?x 7)` instead of `(< ?x 7)`), can be viewed as having a missing literal and an extra literal.

KR-FOCL: A Knowledge Revision Tool

We are advocating that part of the documentation for an expert system consists of a set of correctly classified examples that exercise every clause of every rule of the expert system. For example, for a fault diagnosis system, the examples might consist of mappings between sensor readings and failures; for a medical expert system, the examples might consist of mappings between patient symptoms and diseases (or treatments). These examples are probably best collected incrementally, as each clause is entered.

FOCL is capable of using these examples to learn an operational concept definition in spite of errors in the rule base. However, FOCL does not fix the rule base to correct these errors. KR-FOCL is designed to use a trace of how FOCL learns to suggest revisions to the knowledge base. The following information is retained for each clause learned:

1. The clauses (if any) of each rule that were operationalized by the explanation-based learning component. We will call these clauses *operationalized clauses*.

2. The variabilized predicates (if any) that were added by the induction component. We will call these predicates *induced predicates*.

As FOCL runs, it records the operationalized clauses and the induced predicates for each clause it creates. We will call those clauses of the domain theory that were never operationalized, *unoperationalized clauses*. Table 4 summarizes this information for the example in Table 3.

KR-FOCL analyzes the trace information to suggest revisions to the knowledge base. There are three alternatives that can occur for each clause that FOCL learns:

Table 4. Trace information used by KR-SaranWrap

Clause	operationalized clauses	induced predicates
	# predicate	
1.	1 no_payment_due	{}
	0 eligible_for_deferment	
	0 eligible_for_armed_forces_deferment	
2.	1 no_payment_due	{}
	2 eligible_for_deferment	
	1 eligible_for_financial_deferment	
3.	1 no_payment_due	{}
	2 eligible_for_deferment	
	0 eligible_for_student_deferment	
	0 enrolled_in_more_than_eleven_units	
4.	1 no_payment_due	{}
	2 eligible_for_deferment	
	1 eligible_for_peace_corp_deferment	
5.	0 no_payment_due	(never_left_school ?v1)
	0 continuously_enrolled	
	0 enrolled_in_more_than_five_units	
6.	{}	(disabled ?v1)
7.	{}	(unemployed ?v1)

unoperationalized clauses:

4 eligible_for_deferment
0 eligible_for_financial_deferment
0 eligible_for_disability_deferment

1. The set of operationalized clauses is non-empty and the set of induced predicates is empty. This is a sign that none of the operationalized clauses used need to be modified to correctly classify the training examples.

2. The set of operationalized clauses is non-empty and the set of induced predicates is non-empty. This may be a sign that one or more of the operationalized clauses need to be modified by adding an extra condition from the induced predicates. These user is prompted to verify one of these possibilities.

3. The set of operationalized clauses is empty and the set of induced predicates is non-empty. This may be a sign that one of the unoperationalized clauses needs to be changed. FOCL searches for clauses in the rule base that meet one of two conditions:

a. The induced predicates subsume an unoperationalized clause. This may be a sign that the clause can be replaced by the induced predicates. In effect, this modification removes superfluous literals from a clause.

b. The induced predicates are subsumed by an unoperationalized clause. This also can be a sign that the clause can be replaced by the induced predicates. In effect, this modification adds extra literals to a clause.

Finally, this condition may also be a sign that the induced predicates may be another clause for the top level predicate or for a predicate that is part of an unoperationalized clause. Note that the induced predicates in FOCL need not be operational, so that the induced predicates can use the existing vocabulary of the knowledge base.

KR-FOCL asks the user to confirm any changes that are suggested. Finally, KR-FOCL suggests deleting unoperationalized clauses. The trace in Table 1 illustrates the types of revisions that KR-FOCL suggests.

The first example in Table 1 is a case in which the operationalized clauses are non-empty and the set of induced predicates are non-empty. This occurred when FOCL was learning the fifth clause (see Table 3). In this clause, FOCL operationalized clauses for `continuously_enrolled` and `enrolled_in_more_than_five_units`. In this situation, FOCL also had to induce a condition, `never_left_school`, to rule out additional negative examples. KR-FOCL asks the user to determine if either of the operationalized clauses should be revised by adding the induced predicate. Note that modifying either clause will result in correct classification of this training set. The choice made by the user will best preserve the structure of the rule base that supports understandable explanations. That is, it makes more sense to say that a student is continuously enrolled if the student has never left school and the student is enrolled in five or more units, than it does to say that a student is enrolled in five or more units if the student has never left school and the number of units in which the student is enrolled is greater than 5.

The second example in Table 1 exemplifies the case when the set of operationalized clauses returned by FOCL is empty (see clause 6 of Table 3). KR-FOCL finds an unoperationalized clause, `(AND (filed_for_bankruptcy ?s) (disabled ?s))`, that is subsumed by the induced predicate, `(disabled ?s)`, and asks the user if the clause can be replaced.

The third example in Table 1 is similar to the second example. However, this time there is no clause that is subsumed by the induced predicate `(unemployed ?v1)` (see clause 7 in Table 1). In this case, KR-FOCL asks if the induced predicate can be a new clause for the top level clause, `no_payment_due`. Although this would result in correct classifications, the user responds negatively. Next, KR-FOCL asks if the induced predicates should be a new clause of any rule for a literal in a clause that was not operationalized. The rationale here is that some clause may not have been operationalized not because the clause itself needs to be modified, but because a rule used to establish one of the conditions of the clause may have another alternative. In this case, clause 4 of `eligible_for_deferment` is unused and KR-FOCL next asks if there should be a new clause for `eligible_for_disability_deferment`. The user responds negatively. Although this would result in correct classifications, it would not make

much sense to say that a student is eligible for a disability deferment if the student is unemployed. Finally, KR-FOCL asks the user about adding an extra clause to any rule that has an unoperationalized clause. The user confirms that `(unemployed ?v1)` is another alternative for `eligible_for_financial_deferment`.

Finally, KR-FOCL asks the user to confirm the deletion of any unoperationalized clauses and warns the user about clauses that use predicates whose definition has changed. The user decides to retain the clause that states a student is eligible for a deferment if the student is eligible for a disability deferment (example 4 in Table 1) and decides to delete the clause that states that a student is eligible for a financial deferment if the student is enrolled at UCI.

Limitations

The current implementation of KR-FOCL has several limitations. First, KR-FOCL does not explain why it recommends a certain revision. We are planning an extension that would address this problem by keeping track of the examples that motivate the recommendation. In its current form, we are assuming that a knowledgeable expert either forgot to mention a condition, or mentioned a superfluous condition. When KR-FOCL suggests a revision, it is expected that the expert will realize that the revision will lead to improved performance. In the future, we hope that by providing an expert with a summary of the examples in which the rules in a knowledge base lead to errors, and a revision which eliminates these errors, KR-FOCL may be used by an expert to help discover the correct rules.

Currently, we have only tested KR-FOCL on small examples with only a few errors. As the size of the rule base and the number of errors increases, it may be necessary to find a way to organize the possible revisions to the knowledge base. For example, instead of serially asking verification questions about a large number of possibilities, it may be better to present the user with a menu of these alternatives. In addition, it might be best to order the revisions to the knowledge base by initially focusing on those that will most affect the accuracy of the expert system. Finally, the capabilities of a program such as SEEK (Politakis & Weiss, 1984) to quickly access the impact of a revision would be useful. Currently, the user must run KR-FOCL again to determine the impact of a change.

When the operationalized clauses and induced predicates are non-empty, KR-FOCL suggests revising every operationalized clause by adding the induced predicates. This typically focuses the revisions to a small portion of the knowledge base. We could further supervise this process by making use of rule models (Davis, 1978) that provide information concerning the classes of predicates that are usually used in clauses that propose certain classes of hypotheses.

Finally, KR-FOCL is limited by the underlying machine learning technology.[4] It cannot operate if FOCL cannot solve the problem by using hill-climbing search, or if the problem cannot easily be expressed by constant-free Horn-Clauses. It cannot operate with rules that require certainty factors, rules that deal with large numbers of constants or real-valued constants[5]. In addition, it cannot operate if the expert system relies on a complex conflict resolution strategy to determine the applicability of rules.

Conclusions

In this paper, we have focused on how machine learning technology, in particular programs that integrate empirical and explanation-based learning, can be used to revise the rule bases of expert systems. It is worth noting that knowledge acquisition technology can also address some problems in machine learning. For example, ETS (Boose, 1984) can help automate the process of selecting a vocabulary for training examples. Many successful inductive programs (e.g., Michalski & Chilausky, 1980; Michie, 1983; Quinlan, 1986) require an expert to determine a set of potentially relevant features. In addition, information such as the rule models used by TEIRESIAS may be useful in constraining the hypotheses considered by a learning program.

Acknowledgements

We'd like to thank Ross Quinlan for his advice on FOIL and Caroline Ehrlich for her comments on an earlier draft of this paper. This research is supported by a National Science Foundation Grant IRI-8908260 and by the University of California, Irvine through an allocation of computer time.

4. Note that KR-FOCL gets around a major limitation of SEEK2 (Ginsberg, 1988), in that KR-FOCL can make use of domain theories expressed as Horn-clauses instead of purely propositional domain theories.

5. Note that if there are few constants, it is practical to add predicates that are true of these constants. For example `(color ? x red)` can be replaced with `(AND (color ?x ?y) (red ?y))`. This is not practical with real numbers.

Bibliography

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

Buchanan, B., Barstow, D., Bechtal, R., Bennet, J., Clancey, W., Kulikowski, C., Mitchell, T., & Waterman, D. (1983). Constructing an expert system. In Hayes-Roth, F., Waterman, D., and Lenat, B., *Building Expert Systems.* London: Addison-Wesley Publishing Company.

Clancey, W. (1984). Classification problem solving. *Proceedings of the National Conference on Artificial Intelligence.* Austin, TX: Morgan Kaufmann.

Davis, R. T. (1978). Model-directed learning of production rules. In D. Waterman & F. Hayes-Roth (Eds.), *Pattern-directed inference systems.* New York: Academic Press.

Klinker, G., Boyd, C., Genetetr, S., & McDermott, J. (1987). A KNACK for knowledge acquisition *Proceedings of the Sixth National Conference on Artificial Intelligence.* Seattle, WA: Morgan Kaufmann.

Ginsberg, A. (1988). *Automatic refinement of expert system knowledge bases.* London: Pittman.

Marcus, S. McDermott, J. & Wang, T. (1985). Knowledge acquisition for constructive systems. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* Los Angeles, CA: Morgan Kaufmann.

Michalski, R. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 2,* 349-361.

Michalski, R. & Chilausky, R. (1980) Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition. *International Journal of Policy Analysis and Information Systems, 4,* 125-161.

Michie, D. (1983). Inductive rule generation in the context of the fifth generation. *Proceedings of the International Machine Learning Workshop. University of Illinois, 65-70.*

Mitchell, T., Keller, R., & Kedar-Cabelli, S. (1986). Explanation-based learning: A unifying view. *Machine Learning, 1.*

Pazzani & Kibler (1990). The utility of knowledge in inductive learning. Technical Report. University of California, Irvine.

Politakis, P & Weiss, S. (1984). Using empirical analysis to refine expert system knowledge Bases. *Artificial Intelligence, 22,* 23-48.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1,* 81--106

Quinlan, J. R. (1990). *Learning logical definitions from relations*. Technical Report, University of Sydney.

Rajamoney, S., & DeJong, G. (1987). The classification, detection and handling of imperfect theory problems. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 205-210). Milan, Italy: Morgan Kaufmann.

Swartout, W.. (1981). Explaining and justifying in expert consulting programs. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence* (pp. 203-208). Vancouver, BC: Morgan Kaufmann.

Appendix I. Rule base of the loan repayment expert system.

Clauses and literals that were deleted to deliberately introduce errors are ~~stricken~~. Clauses and literals that were added are shown in bold.

```
(no_payment_due ?s) IF (continuously_enrolled ?s)
(no_payment_due ?s) IF (eligible_for_deferment ?s)

(continuously_enrolled  ?s) IF (AND ~~(never_left_school ?s)~~
                                    (enrolled_in_more_than_five_units ?s))


(eligible_for_deferment ?s) IF (eligible_for_military_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_peace_corps_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_financial_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_student_deferment ?s)
(eligible_for_deferment ?s) IF (eligible_for_disability_deferment ?s)

(eligible_for_military_deferment ?s) IF (AND (enlist ?s ?a)
                                             (armed_forces ?a))

(eligible_for_peace_corps_deferment ?s) IF (AND (enlist ?s ?a)
                                                (peace_corps ?a))

~~(eligible_for_financial_deferment ?s) IF (unemployed ?s)~~
(eligible_for_financial_deferment) IF (AND (enrolled ?s ?c ?u)
                                           (uci ?c)))
(eligible_for_financial_deferment ?s) IF (filed_for_bankruptcy ?s)

(eligible_for_student_deferment ?s) IF (enrolled_in_more_than_eleven_units ?s)

(eligible_for_disability_deferment ?s) IF (AND (filed_for_bankruptcy ?s)
                                               (disabled ?s))

(never_left_school ?s) IF (AND (longest_absence_from_school ?s ?a) (> 6 ?a))

(enrolled_in_more_than_five_units ?s) IF (AND (enrolled ?s ?school ?units)
                                              (school ?school)
                                              (> ?units 5))

(enrolled_in_more_than_eleven_units ?s) IF (AND (enrolled ?s ?school ?units)
                                                (school ?school)
                                                (> ?units 11))
```

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Design Knowledge Acquisition: Task Analysis and a Partial Implementation

Yoram Reich
Department of Civil Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
email: yoram@cs.cmu.edu

Abstract

Design is conceptualized as an ill-structured process that requires diverse knowledge that is hard to acquire. Systematic analysis of design and the knowledge requirements is has in general and in the context of bridge design shows that the knowledge needed can be semi-automatically acquired by using machine learning techniques. Although there are limitations to the approach, preliminary results in the bridge design domain are promising and can potentially transfer to other design domains.

1 INTRODUCTION

The process of knowledge acquisition for any type of expert system is time-consuming and tedious. This effort increases when dealing with design domains that are ill-structured by their nature. One approach that promises to alleviate the difficulty of the knowledge acquisition process is the introduction of learning into system development and maintenance stages (Reich and Fenves, 1989c; Shalin *et al.*, 1988; Witten and MacDonald, 1988). For tasks that are relatively well understood and specified, the above promise can be realized by using a single learning method. For example, one can use the learning program ID3 (Quinlan, 1986) to create decision trees for classifying symptoms into malady classes. Single learning techniques have also been used for very restricted aspects of design (e.g., knowledge refinement (Mitchell *et al.*, 1985) or the acquisition of simple design rules (Arciszewski *et al.*, 1987)); however, no overall approach for the acquisition of design knowledge has been proposed. The goal of this research is to develop a framework for automating and assisting in design knowledge acquisition.

Understanding the issues involved in design knowledge acquisition requires a systematic analysis of the task, deriving the knowledge it uses, the knowledge dynamics and application

(Gaines, 1987). The requirements formulated by this analysis are then mapped into specific knowledge acquisition methods which either exist or should be developed. These methods can be viewed as *generic tasks* (Chandrasekaran, 1986) for solving learning problems (Reich and Fenves, 1989b). An abstract account of a similar methodology based on system engineering is described by Rook and Croghan (1988).

Two observations from the analysis described later are important to note.

1. *The nature of design process.* Design is conceptualized as an iterative process composed of five sub-tasks executed in sequence: problem analysis, synthesis, analysis, redesign, and evaluation. A framework for design knowledge acquisition should address each of these tasks. Clearly, this view is a simplification of real design, nevertheless, it still constitutes an important hard problem.
2. *The need to integrate diverse knowledge acquisition techniques.* The distinct nature of the five design sub-tasks entails the use of different techniques suitable for each. We identify machine learning techniques that can support the acquisition of knowledge for these tasks, and illustrate their integration. The three types of expertise transfer discussed by Gaines (1987) are used in our approach: synthesis, redesign, and potentially analysis knowledge are acquired *by example*; redesign and potentially evaluation knowledge are acquired *by evaluation*; and redesign knowledge is also acquired as *an apprentice*.

1.1 Paper organization

The next section analyzes the design process to derive its knowledge requirements[1]. Section 3 identifies general learning approaches that can acquire design knowledge based on the above requirements and an analysis of the application domain. The identification of specific learning programs is entirely directed by the analysis of the application domain. Section 4 demonstrates the approach in the domain of bridge design by describing a system under development called Bridger. We concentrate on capturing synthesis and redesign knowledge and briefly illustrate the *optional* capabilities of acquiring analysis and evaluation knowledge. The problem analysis is the least understood task. We do not deal with the acquisition of knowledge for this task in this paper. Section 5 concludes with a summary and future work.

[1] Shalin *et al.* (1988) have provided a formal analysis of several tasks and their relevance to existing machine learning techniques. Their analysis of design is very simple, it does not allow a true appreciation of the task complexity and it prevents a detailed analysis of the prospects of using machine learning for knowledge acquisition. Consequently, they were not able to map any learning program onto design knowledge acquisition.

2 THE DESIGN TASK

Design is conceptualized as a five-stage process (see Figure 1a). We first describe these stages and then presents design as a more complex process composed of several phases. We then limit the scope of the approach and provide an analysis of our target domain: the design of cable-stayed bridges.

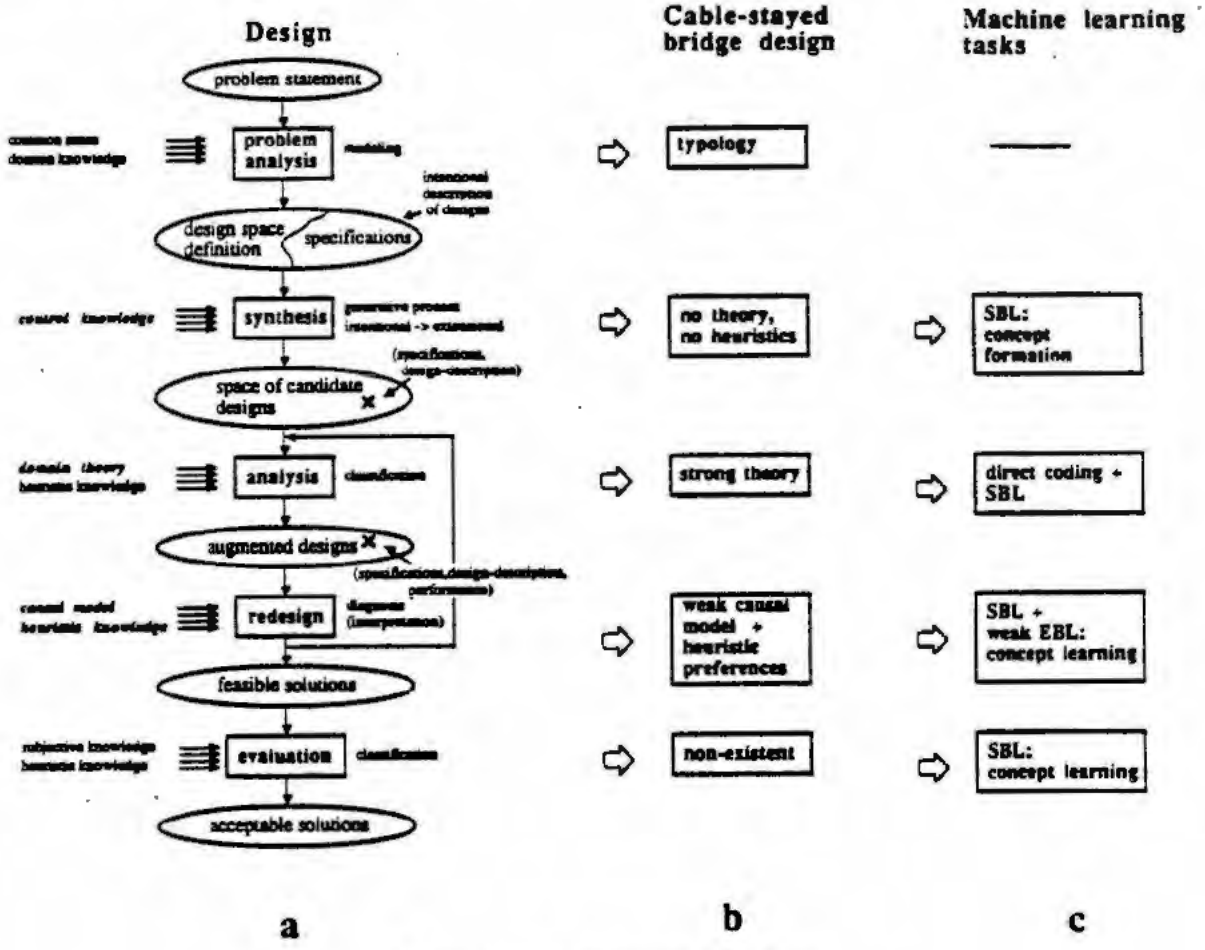


Figure 1: The structure of design

2.1 Design stages

Problem analysis. A design starts with a given problem statement containing some abstract requirements to be achieved. The first design stage is a *problem analysis* of this statement, trying to understand its nature and to define it more precisely. This stage involves the use of common-sense as well as domain specific knowledge. Problem analysis might require an initial

study and possible negotiation with the client who provided the problem to better understand his intentions. Ultimately, this "understanding" results in a detailed set of specifications and a definition of a design search space. This space is also an *intentional* description of all the possible designs.

Synthesis. The second design stage is the *synthesis* of candidate designs that satisfy the specifications. Synthesis uses search techniques for exploring the design space. Control knowledge from diverse sources is used to guide this exploration. Synthesis can also be viewed as a transformation from intentional to extensional description of designs, where the latter is the pair *(specifications, design-description)*.

Analysis and redesign. Since not all the problem constraints are explicitly stated in the problem statement or detected by the problem analysis, and since the design space might be incomplete or contradictory, many candidate designs might be inadequate. Designs might violate hard constraints or be inefficient in using material, space, etc. Candidate designs are analyzed in the third design stage – the *analysis*. Analyses vary in the way they model the artifact. Approximate techniques use course models but can still rely on theory (e.g., beam vs. finite-element analysis of a bridge deck). The analysis or performance calculated is augmented to each design description – now the tuple *(specifications, design-description, performance)*.

Next, *redesign* resolves the artifact inadequacies. Redesign uses heuristic knowledge as well as causal models of the domain. Both suggest modifications and supply preferences for the best possible design modifications. Analysis and redesign may iterate until the design complies with all the requirements and satisfies all the constraints, henceforth called a feasible design.

Evaluation. Feasible designs are evaluated in the fifth design stage – the *evaluation*. In real world problems there will be many feasible solutions that need to be ranked. Ranking designs can be done in several ways. One way of reducing the set of feasible designs is to only keep designs that are Pareto-optimal with respect to some criteria, for example, the selection of designs that have least weight and are cheap to manufacture. Most evaluations, however, rely on heuristics or subjective criteria like esthetics or simplicity. The highest ranked designs are the product of the overall design process.

2.2 Design phases

Most artifacts have levels of details: main components, sub-components, and individual parts. It is not a good practice to design such artifacts in one phase. Rather, the design process should follow the different levels of abstraction of the artifact, leading to a top-down approach composed of several phases: the *conceptual*, the *preliminary*, and the *detailed* design phases.

The five design sub-tasks re-occur in each of the phases, but with a different nature. Problem analysis becomes shorter and produces better defined specifications. Synthesis shifts from overall configuration of components to the generation of parts within these components. Analysis changes from approximate and shallow into exact and detailed. Redesign and evaluation focus on concrete parts that were recently generated and evaluated. Due to the hierarchical nature of this approach, overall backtracking is almost eliminated. The structure of the three phases is similar to one another, therefore we assume the existence of a single phase in the remaining of this paper.

2.3 Scope of design

Our view of design is similar to that of General Design Theory (Yoshikawa, 1981). In this approach, specifications and artifact descriptions are represented as sets in topological spaces. Design is a mapping from the specification topology to the artifact topology. The mapping can be based on heuristic associations or derived in stages by knowledge of the domain. The missing aspect in General Design Theory is that it does not show how such mapping can be acquired.

Heuristic mapping is amenable to preliminary or conceptual design, where major design decisions are made without good quantitative measure of their tradeoffs. In these phases, there is no clear way of structuring the domain to allow for a direct mapping between the goals of the design to the functionality of available components; rather, many artifact properties contribute to achieve the design objectives. In particular, this view is appropriate to our target domain which is cable-stayed bridge design (Reich, 1990b; Reich and Fenves, 1990a).

This approach differs with common approaches to VLSI design where top-down refinement is used to recursively decompose goals into subgoals and assign components that match the subgoals with their functionality (Steinberg, 1987)).

In our design problem we make an assumption that artifacts are represented by lists of property-value pairs. This restricts the scope of design to domains where the topology of artifacts is pre-determined[2].

2.4 Cable-stayed bridges domain

The nature of the domain specific knowledge that needs to be acquired substantially influences the choice of learning methods (see Figure 1b). There is no theory of synthesis in our domain. Moreover, synthesis knowledge is virtually non existent for modern bridge design.

[2]This restriction can be relaxed based on a recent research (Thompson and Langley, 1989).

Books on the subject (Podolny and Scalzi, 1986; Troitsky, 1988) provide a base-line for describing designs, and analyzing them – essentially the product of a simplified problem analysis stage. Practically, catalogues of existing designs are the only source of synthesis knowledge. This entails the construction of knowledge from scratch, leaving machine learning techniques as the primary means for knowledge acquisition beside the tedious process of knowledge engineering with an expert designer.

Analysis of bridges is a well defined procedure in the form of a *finite-element analysis*. It can be easily coded completely. Approximate analyses techniques are also available for manual calculations. Redesign knowledge partially exists as a weak causal model that given some constraint violations, can point to possible design modifications for correcting the problems. In addition, design case studies have been compiled in the past into an unstructured collection of preferences that can be used to resolve conflicts in the causal model. The causal model and the heuristics provide enough substance to build a relatively competent redesign module.

A general statement about design as well as other domains is that knowledge is acquired incrementally by experts. In design, experts need to continuously accommodate new experiences and technologies. Consequently, learning techniques for design knowledge acquisition should be incremental.

3 MAPPING LEARNING ONTO DESIGN

The five design tasks are very different by their nature, utilizing knowledge of different forms that is used by different procedures. This observation is important for identifying the ways in which knowledge for these tasks can be acquired for a specific domain. First, the tasks are mapped into generic artificial intelligence tasks. This can further assist in identifying appropriate machine learning techniques for supporting knowledge acquisition. Next, a specific learning program is assigned to each task based on the characteristics of the application domain.

3.1 Problem analysis

Problem analysis is a process of modeling using diverse and unformalized knowledge. Such a process in general is not captured by any existing artificial intelligence paradigm. We assume that this process is performed manually by a knowledge engineer or a domain expert. This process not only provides a language for describing artifacts and processes in a particular domain, but more importantly, it governs the specific machine learning tools chosen for each learning task identified.

3.2 Synthesis and concept formation

Since bridge synthesis theory does not exist, *similarity-based learning (SBL)* is a candidate methodology for the learning task[3]. Furthermore, statistical learning techniques can be used since many examples of bridges exist (approximately 100) or can be generated and analyzed. Within these limits, the best method for learning knowledge for a generative process is concept formation. The need for concept formation, rather than concept learning is argued next[4].

Suppose there was only one specification governing the design of an imaginary artifact and suppose that there were many possible designs for each specification. The problem of acquiring knowledge for this domain can be transformed into a concept learning task where the possible classes are the possible values of the specification (denoted by v_{ij}) and the examples of a certain class are the candidate designs satisfying the corresponding specification value. One could generate training examples by constructing arbitrary designs and evaluating them to identify which specification they satisfy. This process is illustrated in Figure 2a. In a design scenario, candidates would be generated from the concept description of the specification. The above simplification–designing for a single specification– is never appropriate for real artifacts that are specified by many specifications which are sometimes redundant or conflicting. Therefore this learning method is inadequate.

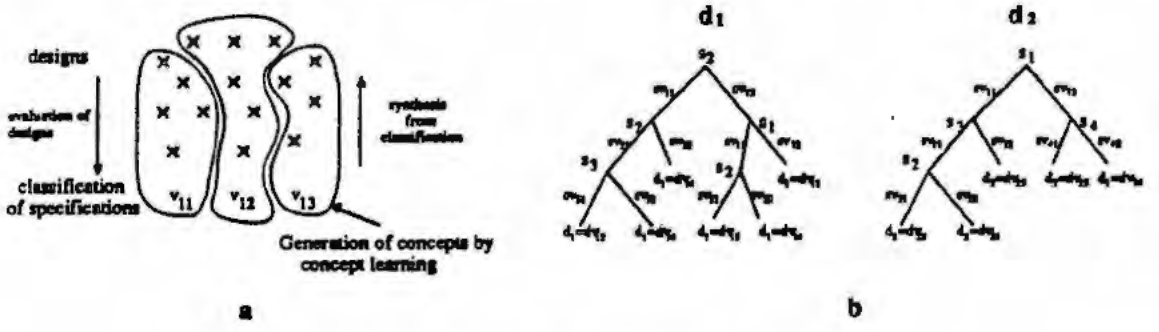


Figure 2: Concept learning and synthesis

An alternative method is using each property of the design as a separate independent design decision that can be carried out by observing the specifications. One way to implement it is to construct a decision tree for each such property (denoted by d_i) where decision nodes represent

[3] In contrast, in VLSI, using decomposition and mapping primitive functions into components (Steinberg, 1987) is a powerful mechanism that with the addition of simulations forms a strong domain theory. This theory allows to conceptualize synthesis knowledge acquisition as knowledge refinement and permits the use of *explanation-based learning (EBL)* techniques (Mitchell *et al.*, 1985).

[4] An elaborate discussion on this issue appears in (Reich and Fenves, 1990b).

different specification properties. A design scenario in this approach consists of using each tree to select a single design description property based on the specifications. This approach is illustrated in Figure 2b. The main problem of this approach is that information implicit in design examples by virtue of describing a single artifact is lost when each artifact descriptor is selected independently. In many cases this indispensable information represents the product of a complex constraint satisfaction process.

The reason underlying the inadequacy of concept learning techniques is the nature of synthesis. Synthesis is a *many-to-many* mapping (e.g., specifications to design description properties). This mapping cannot be divided into several *many-to-one* mappings to be handled by concept learning since design description properties are not mutually independent. Furthermore, synthesis cannot be modeled by a mapping from specifications to classes of designs since such classes are not known a priori, except for some types of routine design.

A method that can capture a many-to-many mapping is concept formation. The approach to synthesis knowledge acquisition by hierarchical, possibly overlapping, concept formation is described in Figure 3. The main idea is that specification and solutions are correlated together and forming classes with their intentional description allows for an appropriate generation of solutions given similar specifications. Ideally, one would form concepts from the final acceptable designs, reducing the need to perform evaluation and redesign in the future. Since the concept formation process is performing an inductive leap (overgeneralization) over the data and cannot guarantee the generation of perfect designs for new specifications, evaluation and redesign stages cannot be eliminated.

There is another dimension to synthesis that constrains the selection of the concept formation method. Namely, the output of synthesis is rarely one alternative; usually, several candidates are produced. The concept formation method used should use knowledge representation that supports such generation.



a. Generation of concepts by concept formation

b. Synthesis

Figure 3: Concept formation and synthesis

3.3 Analysis and concept learning

A single analysis is a process of observing an artifact and stating whether it conforms to some criterion. A complete analysis consists of many such checks and consequently can be conceptualized as a set of classification processes[5]. Exact analysis is relatively easy to capture by direct coding. The ability to perform heuristic analysis can be acquired by generalizing over exact, tedious analyses. Each analysis can serve as a training example where the concept class is the analysis result. Given enough experience, a new artifact can be heuristically classified.

Heuristic analysis never replaces exact analysis, nevertheless, it is useful in many of the iterations of the design process when course measures are sufficient to make decisions. In the last iterations, exact analysis is used to verify the final design. Figure 4 summarizes analysis knowledge acquisition.

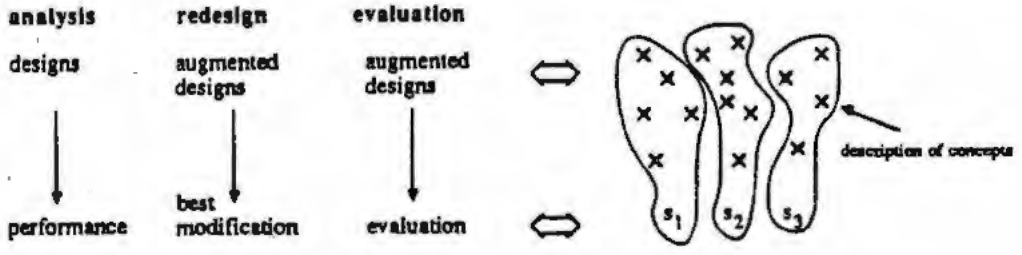


Figure 4: Concept learning for analysis, redesign, and evaluation

3.4 Redesign and concept learning

Redesign is a process of identifying the causes (e.g., design decisions or artifact descriptors) for a specific behavior of the design and determining the changes that these causes should undergo to correct the behavior. This is a diagnosis task augmented by the ability to alter the causes for correcting the inappropriate behavior.

Heuristic knowledge for diagnosis can be captured by a concept learning process (see Figure 4). For satisfying our purpose, examples should consist of a list of violated constraints and the concept classes should be the possible modifications. In some domains (e.g., our target domain), causal models can be acquired as well. This can be done by performing a sensitivity analysis of artifacts. In general, this process will be costly and should be avoided. It is hoped that coding such causal models with the help of an appropriate knowledge elicitation tool is still a reasonable task compared to acquiring general heuristic knowledge from an expert.

[5]Analysis is also a many-to-many function (e.g., design description to performance properties); but it can be modeled by several many-to-one functions.

Note that acquiring diagnosis knowledge by concept learning assumes that there are only small interactions between candidate modifications; otherwise, concept formation is more suitable for gradually assimilating the interactions as it does in capturing synthesis knowledge.

3.5 Evaluation and concept learning

Evaluation is a heuristic and subjective classification of an artifact. Acquiring such knowledge can be done in an apprentice mode or by extracting implicit judgment from existing designs. In our approach, part of the evaluation knowledge is captured as synthesis knowledge. We do not make any further attempt to acquire subjective evaluation, although such acquisition can be performed by a concept learning task in a similar way to the acquisition of heuristic analysis knowledge (see Figure 4).

Figure 1c summarizes the mapping between design tasks onto machine learning tasks for the domain of cable-stayed bridge design.

4 BRIDGER AS A MODEL OF LEARNING FOR DESIGN

Bridger is a learning system for knowledge acquisition and performance improvement that is currently under development. In its intended domain – the design of cable-stayed bridges – it is expected to perform a detailed preliminary design given a set of specifications. The problem analysis stage is compiled into a language for describing specifications, constraints, design examples, and analysis methods. Bridger is not fully integrated: the synthesis and evaluation modules are implemented and tested and the redesign module is in its initial stage of implementation. In the next section the name Bridger refers to the synthesis module only.

4.1 Supporting synthesis by concept formation

Bridger is built on the foundations of the learning program COBWEB (Fisher, 1987)[6]. It extends COBWEB along several dimensions. Bridger can handle entities described by a combination of nominal and continuous property types; it has a correcting-hierarchy module; it has a richer set of learning operators; it can forget undesired knowledge; it can perform directed experimentation that increases the utility of its knowledge; and it can also perform a simple form of constructive induction. Such extensions can improve the ability to master design rather

[6]The decision to choose a COBWEB-like mechanism as the underlying concept formation process for learning synthesis knowledge is the product of the problem analysis stage which determined that for the design of bridges, specifications, artifact descriptions, and even evaluations can be represented by a list of property-value pairs.

than simple diagnostic domains[7].

The learning approach. Bridger uses an incremental learning scheme for the creation of hierarchical classification trees. Bridger accepts a stream of designs described by a list of property-value pairs and builds a classification hierarchy in the following way. When a new design is introduced, Bridger tries to accommodate it into the existing hierarchy starting from its root. Bridger can perform one of several operators to accommodate the new design into the hierarchy and/or modify the hierarchy. If the design has been accommodated into an existing sub-class (i.e., using one of the learning operators), the process recurses with this class as the root of a new sub-tree. Bridger uses a category utility function (Gluck and Corter, 1985) to select between its available operators. The best operator is the one that results in a new classification that maximizes the category utility function.

Synthesis scenario. Bridger synthesizes using a mechanism similar to learning. Bridger sorts the new specification through the tree to find the best host node for the new specification. Synthesis progresses by assigning the new artifact characteristic property-value pairs describing the nodes traversed (see example in the next section). Characteristics of a node are represented by property values that describe most of the designs in the node and almost none in the other nodes. This strategy, which is different than the original method employed by COBWEB, can be viewed as a least-commitment, top-down refinement strategy (Reich, 1990b).

We now illustrate how Bridger uses a hierarchy of bridges to synthesize a new bridge in the domain of Pittsburgh bridges[8]. In this domain, a bridge is described with 12 properties of which seven are specification properties (three continuous and four nominal): the river and exact location of the bridge, the period it was constructed, the purpose of the bridge, the number of lanes and length of the bridge, and whether a vertical clearance requirement was enforced in the design. Five design description properties are provided for each design: the material used, the span of the bridge, the type of the bridge, the location of the road with respect to the bridge, and the relative size of the span to the channel width. Figure 5 illustrates some of these properties.

Figure 6 shows the hierarchy generated by Bridger from 60 examples. The nodes at the hierarchy are described using their characteristics only. The reference symbol and the number of examples below each node (in parentheses) are also provided. Bridger is required to synthesize a *highway* bridge on one of the rivers where *vertical clearance governs*. Bridger starts designing by classifying the design with the hierarchy. The current state of Bridger's

[7]For additional details on Bridger, including statistical performance tests on other domains, see (Reich, 1989; Reich and Fenves, 1989a; Reich, 1990b; Reich, 1990a).

[8]This domain is different that our target domain, but is similar in nature.

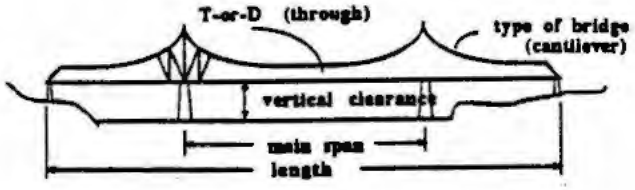


Figure 5: A partial description of a bridge

knowledge is knowing how to design *through* bridges. This property-value is assigned to the new design. Since vertical clearance governs the design, the bridge description should be refined using class G130. The new design will be of a *simple-truss* configuration made of *steel*. Since G130 consists mainly of bridges with 2 lanes, this specification property refines the bridge specification of the new design. Class G138 is chosen next since it represents the *highway* bridges. The overall length of the bridge (1088 ft) also refines the bridge specifications. At this stage the design terminates since all the specifications have been met. The resulting bridge is partially specified since the main span and the relative length of the bridge are not determined. This is expected since the specifications are abstract. Bridger can use other refinement strategies to complete the synthesis[9] or maintain the abstract design as the solution (Reich, 1990b). Furthermore, Bridger can generate a ranked set of alternatives from the subtree whose root is G138 – a necessary requirement of synthesis.

4.2 Supporting analysis by direct coding and concept learning

Analysis of bridges is performed by a finite-element procedure. This analysis can be viewed as a perfect domain theory for this task. The results of the analysis are compared with design codes, that are compiled, mandatory limits on behavioral functions, for identifying the critical parts of the design. A part of analysis which identifies the critical loads on a bridge (i.e., calculates influence lines) is seen in Figure 7. We do not make any attempt to compile analysis results into heuristic analysis because we expect that the synthesis module will absorb some of these heuristics when successful, acceptable designs are learned by Bridger.

Although we do not intend to use heuristic analysis knowledge, we performed an experiment to check the potential of this approach. Five hundred designs of cable-stayed bridges were randomly generated and analyzed by the finite-element procedure. Examples were generated by concatenating the design description with the analysis results of each design. In the experiment, examples were gradually used to generate a classification hierarchy with the same technique

[9]This provides a mechanism for performing exact analysis of abstract design.

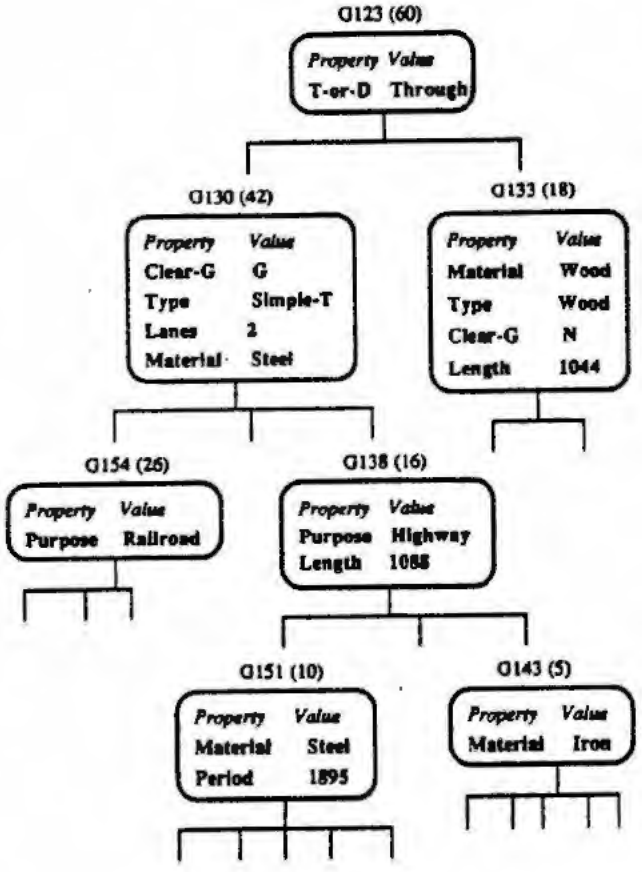


Figure 6: Design concepts hierarchy after learning 60 examples

that learns synthesis knowledge. The hierarchy was used to heuristically analyze the remaining examples based only on their design description part. Each heuristic analysis was checked against the actual finite-element results for the particular bridge. The following *average* results were obtained. The heuristic results continuously improve when additional examples are learned. Improvements range from a match of 1000% after learning 100 examples to a match of 30% after learning 400 examples. Four out of twelve of the results of the heuristic analysis converge to within 8% from the exact values after learning 400 examples. One result remains far from the exact analysis (200% mean relative error), but this is caused by only a small number of major errors. We view this experiment as supporting the claim that heuristic analysis can be learned as well.

Bridger's ability to augment synthesis with analysis knowledge is also the reason for not trying to learn evaluation knowledge separately from synthesis. Evaluation is used as a means to obtain the feedback on the designs produced by the system. The feedback is then used to

enhance synthesis. An example of an aesthetic evaluation that can be augmented to synthesis is: avoid generating bridges that have two pylons or more, with pylon height more than one-forth of the main span.

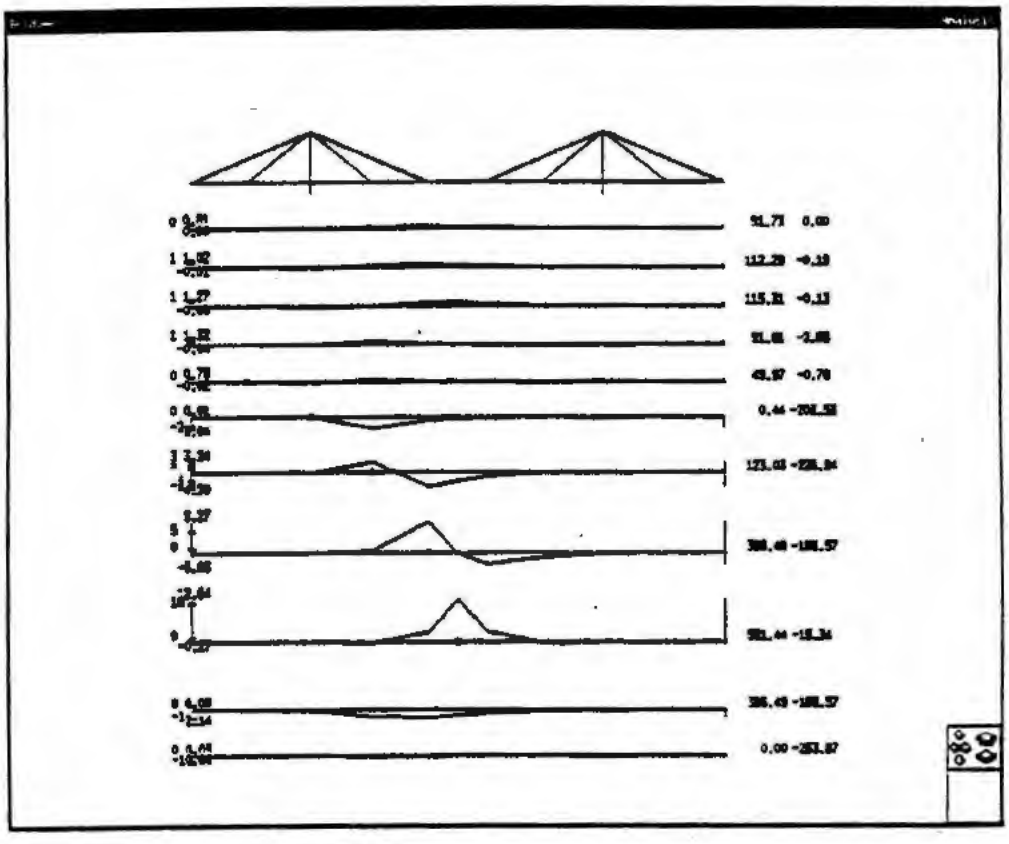


Figure 7: Theory-driven evaluation

4.3 Supporting redesign by concept learning

A simple causal model containing redesign knowledge in the form of qualitative relations is shown in Figure 8. A plus sign on an arrow denotes a positive influence of the tail on the head whereas a negative sign denotes the opposite. Such a graph can be generated systematically for our domain, although it might be incomplete. Note that the graph has two problems: (1) conflicting paths between *cable-spacing* and *bridge-stiffness* and (2) conflicting paths between *pylon-height* and *bridge-stiffness*. Such conflicts can be resolved by by performing a sensitivity analysis for the modification in question or by using experience (e.g., increasing pylon-height leads to an increase of bridge-stiffness). Sensitivity analysis is an expensive procedure compared to the use of experience. Consequently, we try to acquire and organize experience for the redesign task of Bridger.

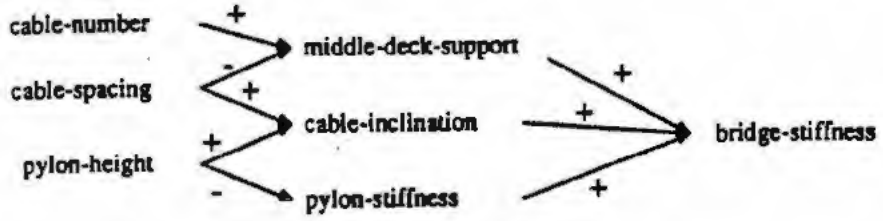


Figure 8: A partial causal model

Two of the main tasks of learning are: (1) the acquisition of heuristics that support the identification of the *best* modification which is usually indeterminate from the causal model, and (2) the refinement of the causal model. At first, such heuristics can be found in technical references. Learning can be used to structure them and resolve conflicts that exists between heuristics from different references. A *case-based reasoning* scheme suitable for this task has been proposed by Clark (1988).

Some of the assumptions made by Clark are not appropriate for our domain, particularly the assumption that conflicts that are once resolved in a certain way, will be similarly resolved in the future. In our domain, heuristics abstract many details to become only approximations; thus, resolving conflicts depends on the context of the specific bridge design scenario. Consequently, learning must be used continuously to further refine the heuristics.

Another system that does not make the above assumption but has a similar case-based representation and reasoning is the Protos system (Bareiss, 1989; Porter *et al.*, 1990). Protos is a system for the acquisition of concepts and their use in future classifications. Protos makes use of an *exemplar-based* representation of categories. In classification, the features of a new case remind Protos, in a process of heuristic matching, of the most similar exemplar in the category structure. The category of the new case is the category of this similar exemplar.

In the cable-stayed bridge domain, categories are possible design modifications, for example, modify the pylon height. Essentially, each design descriptor is a category name since its modification can influence the performance. Other categories can be generated for modifications of design description properties that are tightly coupled. For example, the category *increase the number of cables while reducing their spacing* is a specialization (i.e., conjunction) of the categories increase number of cables and reduce cable spacing. Exemplars in the category structure are abstract bridge descriptions with their performance. Figure 9 depicts an abstract partial category structure consisting of three categories and two exemplars.

In redesign, the performance properties of a candidate design (especially those that violate constraints) serve as features for accessing an existing exemplar in the category structure. The

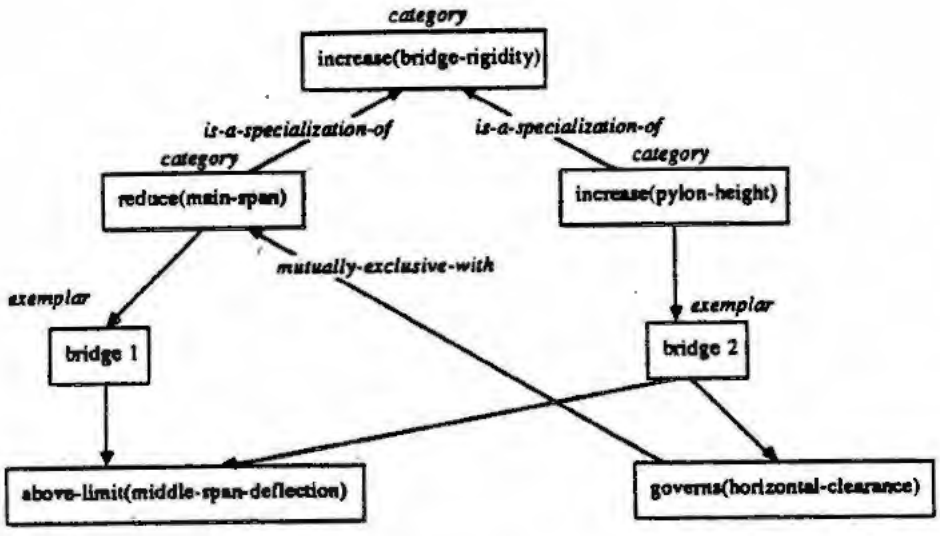


Figure 9: An abstract category structure

category of the chosen exemplar is the best redesign modification. For example, bridge 1 in Figure 9 will be most likely chosen as a match for a new case whose middle span deflection exceeds the allowable limit, unless the horizontal clearance requirement does not allow it.

Bridger analysis module can provide immediate feedback on the redesign performance. If all the violated constraints of the design have been eliminated, the redesign was successful. Otherwise, a relative measure of the improvement can be calculated and stored in exemplars to improve future indexing. This might entail a modification to the hypothesis formation of Protos in that remindings to exemplars will be scaled by the utility stored in them.

The use of Protos is based on the hypothesis that a reasonable causal model without many conflicts can be constructed. Existing conflicting paths, such as those described before, can be resolved by exploration and refinement. This hypothesis remains to be confirmed by testing the complete Bridger system. If it is refuted then the redesign can be augmented by a concept formation system similar to the one used for learning synthesis knowledge (or the one experimented with learning heuristic analysis knowledge). In redesign the concept formation system will be consulted first to retrieve the best fix of the design. This suggestion can then be confirmed by the Protos system. The concept formation system will gradually absorb the knowledge implicit in the apprentice system and in redesign experiences. We do not start with using concept formation since it requires many examples and do not currently use the available redesign knowledge.

5 CONCLUSIONS AND FUTURE WORK

Design is a complex activity. Structuring it and understanding the knowledge it requires for performing a particular task, allows to define specifications of learning techniques that can acquire the domain knowledge needed (see Figure 10). Taking account of the specific knowledge sources available in the application domain allows to select specific machine learning programs to match the above specifications. A partial implementation, called Bridger, exemplifies this methodology in the domain of bridge design.

Since Bridger's modules integrate performance with continuous learning, machine learning techniques are used for all knowledge acquisition aspects possible (Shalin *et al.*, 1988): initial system construction, refining of the existing knowledge base, and adaptation of the knowledge base to a specific style.

The methodology presented is suitable for other design domains. When following the steps in Figure 10 for different domains, different machine learning methods will emerge depending on the particular characteristics of these domains.

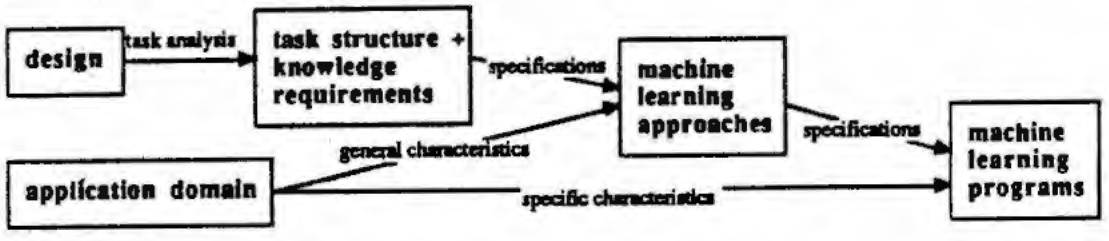


Figure 10: A summary of overall methodology

Future work includes completing the implementation and fully integrating the design modules. As we do not anticipate the full automation of design, we intend to provide a full interactive capability between the designer and Bridger. The designer will be able to direct learning and the use of knowledge in the synthesis module (e.g., select the learning operators manually). The designer will be able to monitor the redesign process and modify and refine the redesign knowledge. Finally, the designer will evaluate feasible designs and submit the best as training examples to Bridger.

Another work involves the extension of the description language to more complex artifact descriptions, especially descriptions with varying topologies. This will allow using the approach in additional types of domains such as mechanical and layout design.

The life cycle of artifacts continues long after the design is completed. Conceivably, the five-stage model of design can be extended beyond the evaluation stage to include the

manufacturing task and the performance of the artifact itself. Consequently, feedback on actual performance of the artifact can be used to acquire evaluation, redesign, and synthesis knowledge. This allows viewing design as supporting the complete life-cycle of artifacts.

Finally, it is useful if Bridger will learn while not used by a designer. Currently, Bridger can heuristically select problems that are expected to enhance its synthesis knowledge. Completing the cycle to analysis, redesign, and heuristic (although not subjective) evaluation, will allow for continuous improvement of knowledge.

Acknowledgments

This work has supported in part by the Engineering Design Research Center, a National Science Foundation Engineering Research Center, and the Sun Company Grant for Engineering Design Research. I would like to thank Prof. Steven J. Fenves for his support and invaluable discussions on this research and to Julian Winsor for his comments on a draft of this paper.

References

Arciszewski, T., Mustafa, M., and Ziarko, W. (1987). A methodology of design knowledge acquisition for use in learning expert systems. *International Journal of Man-Machine Studies, 27(1)*, 23–32.

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition.* Academic Press, Boston, MA.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert, 1(3)*, 23–30.

Clark, P. (1988). Representing arguments as background knowledge for constraining generalisation. In Sleeman, D., editor, *Procedings of the Third European Working Session on Learning*, pages 37–44, Pitman, Aberdeen.

Fisher, D. H. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2(7)*, 139–172.

Gaines, B. R. (1987). An overview of knowledge-acquisition and transfer. *International Journal of Man-Machine Studies, 26(4)*, 453–472.

Gluck, M. and Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–287, Academic Press, Irvine, CA.

Mitchell, T., Mahadevan, S., and Steinberg, L. (1985). LEAP: A learning apprentice for VLSI design. In *Proceedings of The Ninth International Joint Conference on Artificial Intelligence*, pages 573–580, Morgan Kaufmann, Los Angeles, CA.

Podolny, W. and Scalzi, J. B. (1986). *Construction and Design of Cable-Stayed Bridges*. John Wiley and Sons, New York, second edition.

Porter, B. W., Bareiss, R., and Holte, R. C. (1990). Concept learning and heuristic classification in weak-theory domains. *Artificial Intelligence, (to appear)*.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1(1)*, 81–106.

Reich, Y. (1989). *Combining Nominal and Continuous Properties in an Incremental Learning System for Design*. Technical Report EDRC-12-33-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Reich, Y. (1990a). Constructive induction by incremental concept formation. In *Proceedings of the Seventh Israeli Symposium on Artificial Intelligence and Computer Vision*, Elsevier Science Publishers, Tel-Aviv, Israel. (in press).

Reich, Y. (1990b). Converging to "Ideal" design knowledge by learning. In Fitzhorn, P. A., editor, *Proceedings of The First International Workshop on Formal Methods in Engineering Design*, pages 330–349, Colorado Springs, Colorado.

Reich, Y. and Fenves, S. J. (1989a). Incremental learning for capturing design experience. Submitted for publication.

Reich, Y. and Fenves, S. J. (1989b). *Integration of Generic Learning Tasks*. Technical Report EDRC 12-28-89, Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA.

Reich, Y. and Fenves, S. J. (1989c). The potential of machine learning techniques for expert systems. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, 3(3)*, 175–193.

Reich, Y. and Fenves, S. J. (1990a). Automatic design knowledge acquisition by concept formation. In Arciszewski, T., editor, *Knowledge Acquisition in Civil Engineering*, American Society of Civil Engineers, New York. (in press).

Reich, Y. and Fenves, S. J. (1990b). The formation and use of abstract concept in design. In Fisher, D. and Pazzani, M., editors, *Computational Approaches to Concept Formation*, Morgan Kaufmann, Los Altos, CA. (in press).

Rook, F. W. and Croghan, J. W. (1988). Knowledge acquisition for AI systems requirements analysis: A system engineering methdology. In *Proceedings of The First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, pages 798–804, ACM, Tullahoma, Tennessee.

Shalin, V. L., Wisniewski, E. J., Levi, K. R., and Scott, P. D. (1988). A formal analysis of machine learning systems for knowledge acquisition. *International Journal of Man-Machine Studies, 29(4)*, 429–446.

Steinberg, L. I. (1987). Design as refinement plus constraint propagation: The VEXED experience. In *Proceedings of AAAI-87*, pages 830–835, Morgan Kaufmann, Seattle, Washington.

Thompson, K. and Langley, P. (1989). Incremental concept formation with coposite objects. In Segre, A. M., editor, *Proceedings of the Sixth International Workshop on Machine Learning*, pages 371–374, Morgan Kaufmann, Ithaca, NY.

Troitsky, M. S. (1988). *Cable-Stayed Bridges: An Approach to Modern Bridge Design.* Van Nostrand Reinhold, New York, second edition.

Witten, I. H. and MacDonald, B. A. (1988). Using concept learning for knowledge acquisition. *International Journal of Man-Machine Studies, 29(2)*, 171–196.

Yoshikawa, H. (1981). General Design Theory and a CAD system. In Sata, T. and Warman, E., editors, *Man-Machine Communication In CAD/CAM, Proceedings of the IFIP WG5.2-5.3 Working Conference*, pages 35–57, North-Holland, Amsterdam.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Automatic Knowledge Acquisition from Texts: Learning Terminological Knowledge via Text Understanding and Inductive Generalization

Ulrich Reimer
Universitaet Konstanz
Informationswissenschaft
Postfach 5560
D-7750 Konstanz, F.R.G.
reimer@inf-wiss.ivp.uni-konstanz.dbp.de

Abstract

An overview of the wit system currently under development at the University of Constance is given. wit understands technical texts and builds up representations of the concepts described therein. It inductively generalizes the concepts learned from the texts to more generic ones. By this means the system automatically acquires terminological knowledge about a domain of discourse. Due to a bootstrapping approach only small domain-specific world knowledge is needed by wit to begin its operation. As not only the representations of new concepts are derived from the texts but also their class memberships, a learning from examples is performed. This ensures that only relevant and no artificial concepts will be learned.

1. Introduction

Today, many artificial intelligence (ai) systems rely heavily upon large amounts of knowledge so that the provision of domain-specific knowledge to an ai system has become crucial to its functionality: This is called the knowledge acquisition bottleneck. To support the process of knowledge acquisition tools have been invented which automate parts of the task (see e.g. (Morik 1989; Musen 1989)). They perform validation checks, detect knowledge gaps and conflicts and map the bits and pieces of knowledge obtained from an expert into an appropriate representation format. Sometimes techniques of machine learning are employed to automate a main portion of the task, e.g., to determine rules to be used for problem solving (Buchanan, Mitchell 1978; Michalski, Chilausky 1980). However, the main part of the knowledge acquisition task still remains manual.

At the Department of Information Science of the University of Constance we are working towards an automatic acquisition of the terminological knowledge of a domain of discourse by way of text understanding. Taking texts as a knowledge source is a quite obvious choice.

The system we are developing (called **wit**) aims at analyzing technical texts which describe special concepts of the domain of discourse to be learned about (e.g. technical devices or chemical substances). There are two reasons why we do not deal with introductory texts or text books. First of all, written material of this kind is not available for all domains of discourse, or is usually out of date (especially for rapidly evolving domains, like information technology). Secondly, the parsing machinery needed to adequately cope with introductory texts for the purpose of knowledge acquisition is worldwide not in sight (despite the success made in this area, see e.g. (Riesbeck 1982; Hahn 1989)).

The approach we have adopted in **wit** differs considerably from previous work on extracting knowledge from texts. So it must be distinguished from work on acquiring lexical knowledge from texts, as e.g. described in (Zernik, Dyer 1987; Antonacci et al. 1989; Slator 1989). The new lexicon entries these systems generate are to be used by a natural language parser and thus must reflect syntactical surface phenomena. This is in contrast to **wit** that acquires knowledge about a domain of discourse, which has clearly to be independent from any textual surface phenomena. (Möller 1988) presents a system to support a knowledge engineer in extracting knowledge encoded in texts. The extraction process is semi-automatic and neither a full-fledged parsing nor a concept generalization is performed. An approach to learning event schemata from text is described in (Riesbeck 1981). Whenever a statement is encountered in a text which contradicts an event schema a representation of that exception is formed. Several exception descriptions of an event schema that can be generalized to a class representation are used to modify the event schema they are exceptions of. An approach to derive rules for a rule-based expert system from texts is described in (Petrie 1985). It suggests a straightforward conversion from if-then-sentences in a text to rules in a production rule language. Thus its applicability is very small, although it may be useful in some cases.

In some aspects similar to **wit** is the work described in (Lebowitz 1983) (IPP) and (Lebowitz 1988) (RESEARCHER). IPP parses narrative texts and generalizes the various event representations built up to prototypical event schemata. Using these schemata for subsequent text parsing enhances the depth and the accuracy of IPP's understanding capabilities. The RESEARCHER system is similar to IPP but generalizes physical object descriptions instead of events. The generalized descriptions are used for disambiguation and thus enhance subsequent parsing, too. The main difference to **wit** with respect to the generalization performed is that **wit** does no conceptual clustering like IPP and RESEARCHER (Lebowitz 1987) but a *learning from examples* because the class memberships of the (example) concepts acquired from the texts are always given. Due to the learning from examples performed by **wit** it is guaranteed that no artificial concepts will be learned, i.e., no concepts which are not part of a human domain model. This is in contrast to IPP and RESEARCHER where the concept classes derived are usually artificial. This does not matter because these concept classes are used for disambiguation, but it would be unacceptable for the purpose of **wit**.

Contrary to wit, most of the current learning systems make a strict distinction between the background knowledge used for learning and the knowledge being learned (but see e.g. BLIP: (Wrobel 1989)). The only knowledge which stays unchanged in wit is the linguistic knowledge[1] whereas the domain-specific world knowledge is not only used as background knowledge for learning but is also knowledge being learned.

This paper gives an overview of our wit system (cf. Fig.1). We only sketch the functionality of the text parser (which is based on the work described in (Reimer, Hahn 1988; Hahn 1989) but elaborate more deeply on the learning component. Sec. 2 deals with transforming texts into a representation of their contents. Sec. 3 is concerned with the inductive generalization from the concepts acquired and outlines how more general concepts of a domain of discourse are learned.

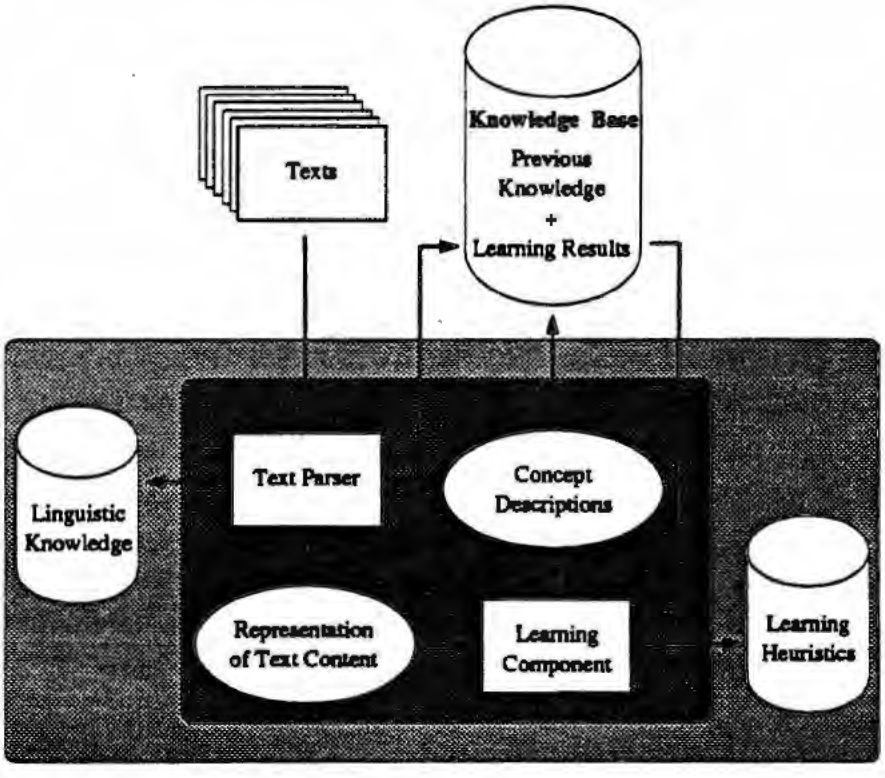


Figure 1: The System Architecture of wit

[1] However, a word that is identified as a previously unknown noun by the text parser will be added to the lexicon.

2. Transforming Texts into a Representation of their Contents

2.1 The Functionality of the Text Parser and its Output

The source from which new knowledge is to be learned by the wit system are technical texts about new data processing products. Among the different kinds of knowledge which may be encoded in a text we aim at the acquisition of *terminological knowledge*, i.e., knowledge about the (actual and possible) properties of a concept and about relationships between concepts. The parsing machinery necessary to extract such kind of knowledge need only support a comparatively small number of syntactical phenomena. Most important are predication ("the DeskWriter *is* an ink jet printer"), attribution ("*disposable* ink cartridge"), and text coherence patterns. Textual coherence means that a text is structurally organized according to certain well-formedness criteria. Exploiting this fact by a parser allows to determine which concepts refer to which other (usually previously mentioned) concepts. The most prevalent coherence pattern is the introduction of a concept (the theme) followed by statements about concepts closely related to it (the rhemes). An example is given by the text shown in Fig.2 (the thematic concept is given in bold and the rhematic ones in italics). We do not want to go into the details of the text parser here and refer the reader to (Hahn 1989), where the predecessor of the parser being developed for wit is described, and to (Sonnenberger 1990).

> "The **DeskWriter** from HP is a new ink-jet printer. *Ink* is deposited from a disposable *ink cartridge* at a *resolution* of 300 dpi. The *cartridges* are priced at a reasonable $18.95. The **DeskWriter** comes with four *fonts*: *Courier*, *Times*, *Helvetica* and *Symbol*."

Figure 2: Illustrative text fragment showing theme-rheme patterns[2]

The concept descriptions built up from a text are usually *incomplete*, firstly because text parsing is in principle partial (Hahn 1989) – especially when only limited world knowledge is available – and secondly because not all properties of a concept that are relevant in a certain domain of discourse may be mentioned in a text. Furthermore, as text parsing is partial some portions of a concept description may be of *uncertain* status. To illustrate the kind of output produced by the text parser of wit Fig.3 shows a representation of the concept 'DeskWriter' as it may be generated (depending on what knowledge the parser has available) from the text given in Fig.2.

More accurately, the parser output is a set of propositions which are added to a knowledge base by a separate system component. This component takes care of modifying the certainty factors, of maintaining alternative parts in a concept representation, and of integrating single propositions into a coherent concept representation (e.g., deciding if a new slot must be created or a slot be filled). The actual wit system operates on German texts, but as its

[2] The text is based on an article in BYTE, Oct. 1989, pp.82–83.

main mechanisms work for English texts as well we give English examples for the reader's convenience.

The knowledge acquired from a text as well as the knowledge generated by the generalization component (see Sec.3) is immediately used for further text analysis (this approach is called closed-loop learning and is also employed in the BLIP system: (Wrobel 1989; Morik, Kietz 1989). This additionally enables the parser to extract knowledge, which, however, may be uncertain or even incorrect since the knowledge used is uncertain and incomplete. The uncertainty and even the incorrectness of some of the freshly learned knowledge is of no harm for the concept hierarchy to be induced (see Sec.3). This robustness is due to the fact that the construction of the concept hierarchy relies on the knowledge obtained from a large number of texts; it also results from the nature of generalization itself, which abstracts away occasionally occurring properties (see Sec.3.3). Thus, uncertainties and inaccuracies do not propagate to the representations of superordinate concepts. Making use of newly learned knowledge in subsequent text analysis allows of what we call 'knowledge base bootstrapping'.

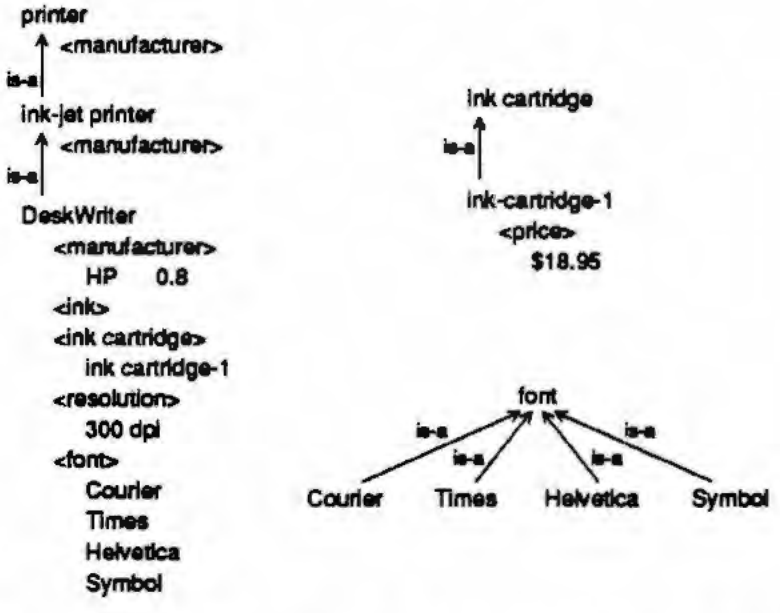


Figure 3: A frame representation as it may result from the analysis of the text given in Fig.2[3]

[3] Slot names are given in angular brackets while slot entries (if any) are given below the slot name and are indented. Numerical values at slots or slot entries indicate certainty factors.

2.2 Bootstrapping a Knowledge Base

Characteristic of the approach we adopt in our wit system is that it starts operation with very small domain-specific knowledge (although it is knowledge-intensive: it makes use of syntactic knowledge, of lexical knowledge, of knowledge about text coherence, and of domain-independent world knowledge, e.g. about part-whole relationships). The knowledge base provided to the system prior to its first operation contains only the names of some very generic concepts of the domain of discourse to be learned about (possibly extended by some slots, like the slot 'manufacturer' in Fig.3; see Sec.3.1). By this means the system knows about the existence of concepts with such names and is able to *focus its attention* to those parts of a text which directly or indirectly (via textual coherence) deal with one of them. Other parts of a text need not be analyzed in depth and are only monitored for certain syntactical phenomena in order to keep track of textual organization. New concepts are created in the knowledge base mainly by three mechanisms:

1. A new concept is created in the concept hierarchy whenever a composite noun of the form $n_1 n_2 \ldots n_m$ is introduced in a text, with n_m being the name of a concept already known. The concept $n_1 n_2 \ldots n_m$ is taken as a new subordinate of the concept n_m. An example is a concept named 'ink cartridge' where $m = 2$ and $n_2 =$ 'cartridge' is a known concept. Especially in technical domains concept names show this pattern.
2. An approach to determine names of concepts which are of a completely different kind than the ones already known (and are therefore not subordinate to them) exploits text coherence patterns. Via such patterns concepts are implicitly associated with other concepts in a text (cf. Fig.2). In the knowledge base such associations are represented by the attachment of slots or slot entries to frames (see Fig.3). Therefore, the names of slots and slot entries of all the frames in a knowledge base are potentially names of yet unknown concepts! A slot or slot entry is finally established as a concept name when a statement about it has been derived from a text. An example of a slot that stands for a concept is the slot 'ink cartridge' in Fig.3 whereas the slot 'resolution' in Fig.3 represents a property only.
3. A third way of deriving knowledge about the existence of new concepts is by comparing the names of all slots and slot entries for common ending nouns[4]. Those slot names (or entry names) which share an ending noun are interpreted as names of new concepts; these concepts are then subordinates of the so far unknown concept named by the common

[4] For composite nouns (but not for adjective/noun pairs) the situation is a bit more difficult in German than it is in English because composite nouns in German are written as one word without hyphenation (e.g. 'Speicher*platine*' (memory *board*) and 'Drucker*platine*' (printer *board*)). To avoid taking suffixes which form no words of their own as concept names the lexical knowledge of the parser is referred to in order to determine if a common ending is in fact a noun. The parser extends its lexicon whenever a word in a text is identified as an unknown noun.

ending noun[5]. For example, from a slot 'cpu board' and a slot 'memory board' it is inferred that they stand for concepts and that they are subsumed by the concept 'board'.

Summarizing, we may say that in the beginning most of the words of a text are nothing more to **wit** than plain character strings, except function words, some general-purpose verbs like 'contain' or 'use', and words that are morphologically reducible to names of concepts in the knowledge base. The last category is used to *focus the attention* of the text parser and provides the starting point to learn new concepts of a domain of discourse.

With respect to having no or only small previously given, domain-specific background knowledge available **wit** is similar to the approach described in (Morik, Kietz 1989). A main difference of the requirements to be met is that the latter approach can assume that its input contains only relevant knowledge whereas **wit** must learn from an input source only some parts of which are relevant to the domain of discourse to be dealt with. Thus, in order to distinguish (potentially) irrelevant information from (potentially) relevant one **wit** must have at least some domain-specific knowledge available when beginning its operation.

3. Acquiring New Concept Representations by Inductive Generalization

3.1 The Basic Approach

The result of running **wit** on several texts is a knowledge base containing representations of the concepts described in the texts and is-a links (also derived from the texts) between these concept representations as well as to superordinate concepts previously known. An example of what kind of knowledge may be derived from one or several texts is illustrated in Fig.4.

Since no concepts can be acquired from a text without some domain knowledge being available the representations of the new concepts learned merge with the concept representations already present. Previously given knowledge is always of a general nature and comprises statements about generic concepts of the domain of discourse. For example, if the concept 'printer' is already known the representation depicted in Fig.4 is to be extended by this concept. By a decomposition of composite nouns and a subsequent substring match as discussed in Sec.2.2 (Case 1) 'ink-jet printer' and 'high-quality printer' will then be considered as subordinate to 'printer'. Due to property inheritance the representations of the freshly learned concepts will then become extended by the slots present at the concept 'printer'. Fig.5 shows this situation and thus extends the simplified, first view underlying Fig.4.

If at least two concepts have been learned that have a common superordinate the generalization component determines their common representation part and integrates it in the representation

[5] Since names of slots can also be names of frames the concept specialization between frames induces an according specialization between slots.

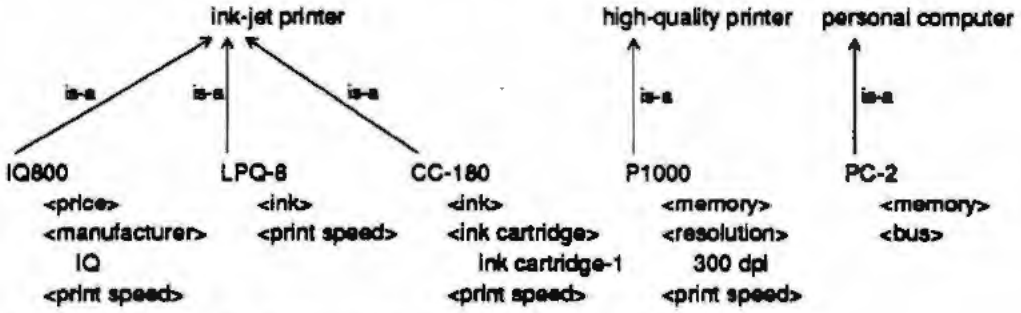


Figure 4: Example of what kind of knowledge wit directly acquires from a text

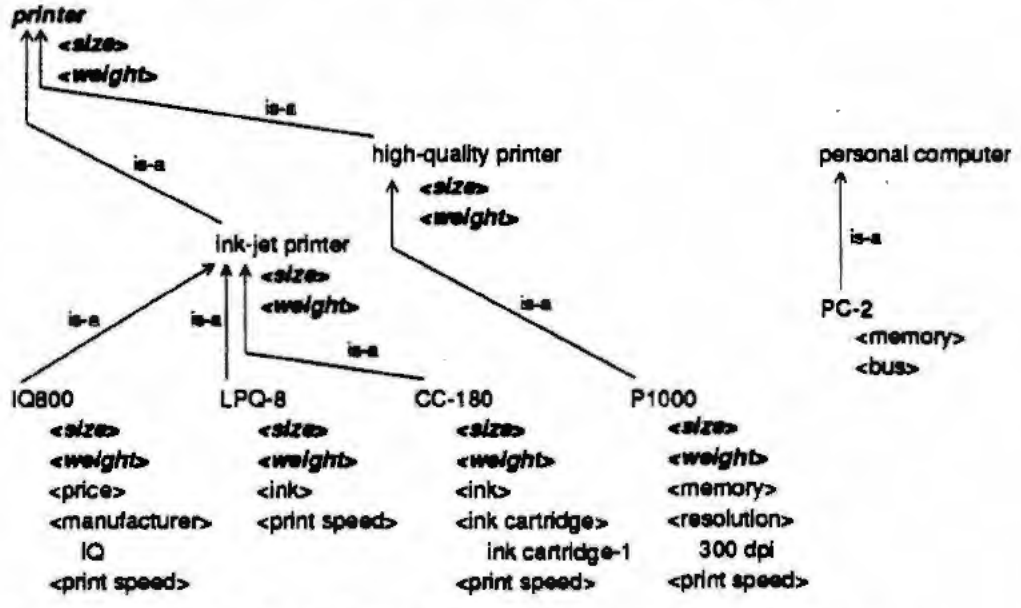


Figure 5: The representation of Fig.4 extended by previously given knowledge (indicated by bold-italics) about the domain of discourse

of the superordinate concept. For the hierarchy depicted in Fig.5 this procedure would e.g. be applied to the concepts 'IQ800', LPQ-8' and 'CC-180' and their superordinate 'ink-jet printer', resulting in the new slot 'print speed' for the representation of 'ink-jet printer'.

Although in principle quite simple, such a concept generalization bears a lot of intricacies. So generalizations that are appropriate will often not be made because the domain knowledge needed to do so is (still) lacking. For example, the two slots 'Helvetica' and 'Courier' cannot be generalized to the slot 'font' unless it is known that both are fonts (as it is the case in Fig.6). In some cases (particularly in technical texts), however, linguistic knowledge about how composite nouns and nominal phrases are to be interpreted permit generalizations even when appropriate domain-specific knowledge is lacking. For example, the two slots 'parallel interface' and 'serial interface' in Fig.6 are at first incomparable if the domain knowledge does not say that both are interfaces. However, linguistic knowledge allows to derive that both

slots refer to subordinates of a concept named 'interface' (cf. Case 3 in Sec.2.2). Therefore these slots can be generalized to the slot 'interface'.

Another problem is that the representations of similar concepts (e.g., all being printers) are usually rather different and largely incomparable to each other at first sight. This is due to the fact that they often stem from different texts which introduce similar (or even the same) concepts differently and deal with different aspects. This is because

- authors only deal with those characteristics of a concept which are considered important by them, i.e., the concept descriptions are partial, and
- different terminology is used, especially when authors have different viewpoints on the same fact.

In order to increase the amount of common features detected for two concept representations domain-dependent background knowledge (see e.g. (Kodratoff, Ganascia 1986, Sec.9.4.2.2)) and the *semantics of the representation constructs* used must be considered. Although the latter requirement is well known (see e.g. (Kodratoff, Ganascia 1986, Sec.9.4.2.1)) it is not deeply examined in the literature. Since this issue is of extraordinary significance to the task at hand we will shortly discuss it in the following section.

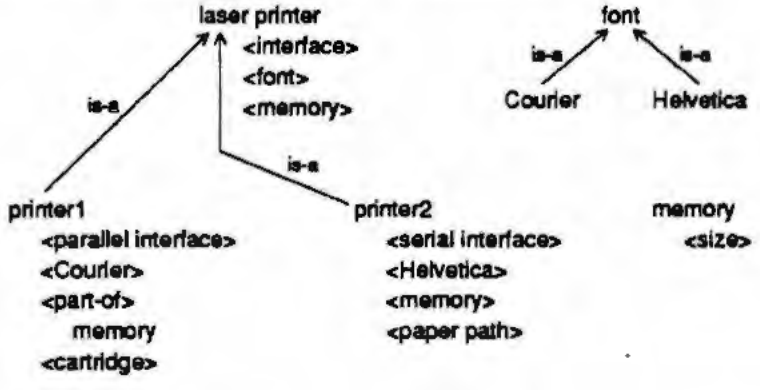


Figure 6: Example of an inductive generalization

3.2 The Semantics of the Representation Constructs Used as Background Knowledge to Generalization

A lot of criteria are known for generating generalizations from a set of given concept representations (see e.g. (Michalski 1983)). A particular generalization algorithm chooses the criteria relevant to the task it is designed for and adapts them to the knowledge representation language used. For a simple representation language (like feature vectors) this is straightforward but if a more complex representation language (like a frame model) is employed severe problems will occur if the concept representations to be generalized are rather inhomogeneous and if the semantics of the representation language used is not adequately accounted for by the generalization algorithm: Many generalizations possible would not be made and

the generalizations constructed would tend to be overgeneralizations. This is a problem to be dealt with in the wit system! To give an illustration consider the two concept representations 'disk drive-1' and 'disk drive-2' as shown in Fig.7. Both frame representations are incompatible if one looks at their structure only. However, if one takes transitivity of the part-of relationship into account it can be deduced that 'disk drive-1' and 'disk drive-2' both consist of a read/write head. Thus, a match of both representations yields as the common structure a part-of relationship from an element of the concept class 'read/write head' to an element of the class 'disk drive'. This gives the justification to construct the generalization shown in Fig.8.



Figure 7: Two structurally different concept representations 'disk drive-1' and 'disk drive-2'

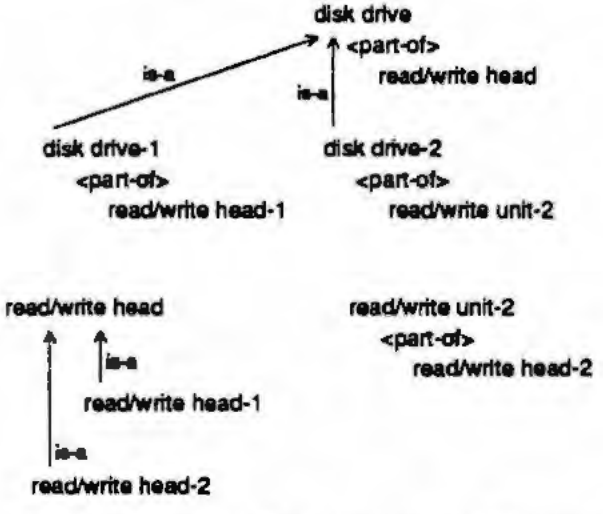


Figure 8: A generalization which is only possible when the transitivity of the part-of relationship is known

The example given in Fig.7 resembles the matching problem discussed in (Lebowitz 1986) (p.210). In contrast to the solution outlined there, which basically suggests to introduce heuristics that allow a greater degree of dissimilarity when matching two representations (and thus to allow invalid matching results in some cases to occur), the problem can be solved

without the need to weaken the exactness of matching by considering the semantics of the representation constructs used.

Similar situations occur if e.g. different slot types have to be generalized. For example, the slot 'part-of' of the frame 'printer1' in Fig.6 (which indicates a part-of relationship between 'memory' and 'printer1') and the slot 'memory' of the frame 'printer2' in Fig.6 (which also indicates a relationship, namely between 'memory' and 'printer2', but of some unspecified kind) have been generalized to the slot 'memory' at the superordinate concept 'laser printer'. That it is correct to do so can again only be deduced from the formal semantics of the frame representation model used (for more details see (Reimer, Mußgnug 1990)).

We can conclude that for the generalization component to generate appropriate generalizations it must know about the semantics of the representation model used. This knowledge is either built into the generalization algorithm – which would lead to a strong dependency of the learning component on the knowledge representation component – or it is provided as meta-knowledge – which would result in a severely inefficient system. A third, more satisfying solution to this problem is suggested in (Reimer, Mußgnug 1990).

3.3 The Generalization Process

A further difficulty with generalization is the need to cope with the occurrence of violations of the semantics of concept specializations, which requires that all parts of a concept representation be present at the representation of all its subordinates (by property inheritance; see e.g. (Reimer 1986)). The reason for these violations to occur is again the fact that for the same kinds of concepts different texts usually mention different features (see Sec.3.1). In wit we deal with this problem by keeping book of the *relative frequency* with which a slot or a slot entry of a concept representation appears at its immediate subordinates (see Fig.9). A slot (or slot entry) of a generalized concept is discarded when its frequency value gets below a cut-off value. Vice versa, a slot (or slot entry) not being part of a generalized concept description but present at some of its subordinate concepts is added to it if its frequency value exceeds the cut-off value (and if one of the conditions (1) and (2) as introduced below is satisfied). We currently use 0.6 as the cut-off value for a generalized slot (see (1) below) and 0.4 otherwise (see (2) below).

Since text parsing is partial so that the text parser sometimes delivers results which are only likely to be correct slots (or slot entries) may be marked as *uncertain*. The determination of relative frequency values must take these certainty factors into account. Thus, the *weighted relative frequency value* of a slot s at a frame f is defined as:

$$f_s(f,s) = \frac{\sum_{f_i \in isub(f)} \overline{cf}(f_i, s)}{|isub(f)|}$$

where $isub(f)$ is the set of all immediate subordinates of the concept f, $|M|$ is the cardinality of the set M, and $\overline{cf}$ is defined as follows:

$$\overline{cf}(f,s) = max(\{cf(f,s_i) \mid s_i \in sub(s) \vee s_i = s\})$$

where $sub(f)$ is the set of all subordinates of the concept f, and

$$cf(f,s) = \begin{cases} 0 & \text{, if } s \text{ is not a slot of } f \\ \text{the certainty factor of slot } s \text{ at frame } f & \text{, else} \end{cases}$$

It is further to be noted that the function sub applied to a slot name that is not a concept name yields the empty set. $\overline{cf}(f,s)$ determines from those slots of f which are equal to s or subordinate to s the one with the highest certainty factor. This is necessary because a frame may possess more than one of these slots (e.g. a slot 'Courier' and a slot 'Helvetica', which are both subordinates of 'font').

Let us give an example. If the association of the concept 'Courier' to 'printer4' as shown in Fig.9 is only probably correct the slot 'Courier' at the frame 'printer4' would be assigned an appropriate certainty factor, e.g. 0.8. In that case the relative frequency value of the slot 'font' at the superordinate 'laser printer' (i.e., f_s('laser printer','font')) would become 0.7 (which is (0.8+1+1)/4); otherwise it is 0.75, as shown in Fig.9[6].

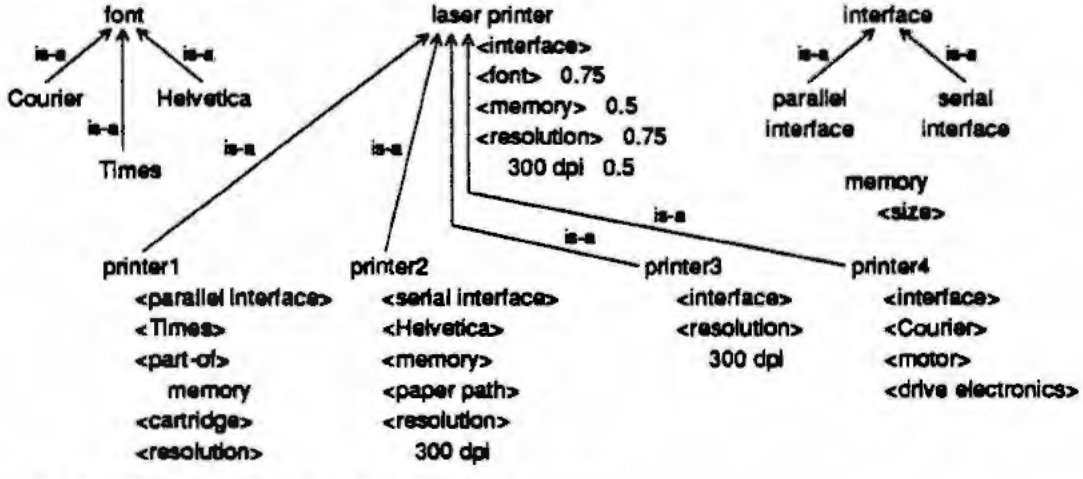


Figure 9: Relative frequency values for slots and slot entries

Now, we can use the measure f_s to define more precisely how a generalized concept description is constructed for a frame g from all its immediate subordinates. The slots of these subordinates are taken as the basis to generalize from; this set is defined as follows:

$$\tilde{S}(g) = \bigcup_{f_i \in isub(g)} slots(f_i)$$

where $slots(f)$ is the set of all slots of a frame f. We now define that a slot s is taken as part of a generalization g if one of the following conditions (1) or (2) is fulfilled.

[6] Certainty factors of 1 are not shown.

Add a generalized slot to g if this is considerably better than adding a more specific slot:

This is expressed by the following condition:

$$s \in S_G(g) \wedge f_s(g,s) \geq c(g,s) \tag{1}$$

where

$$c(g,s) = max(\{0.6\} \cup \{f_s(g,s') + 0.2 \mid s' \in sub(s)\} \cup \\ \cup \{f_s(g,s') \mid s' \in super(s) \cap S_G(g)\})$$

$sub(f)$ is the set of all subordinate concepts of f, and $super(f)$ is the set of all superordinate concepts of f. Moreover,

$$S_G(g) = \left\{s \mid s \in \tilde{S}(g) \wedge \exists s' \in \tilde{S}(g) : s' \in sub(s)\right\} \cup \bigcup_{s' \in S_S(g)} isuper(s')$$

is the set of all (not too much) generalized slots from which can be chosen. It is $isuper(f)$ the set of all immediate superordinate concepts of f, and $S_S(g)$ is the set of the most specific slots in $\tilde{S}(g)$ as defined below. In verbal terms condition (1) means that a generalized slot is only then taken as part of a generalized concept description g if its weighted frequency value is at least the maximum of the frequency values of all competitive slots (including the cut-off value of 0.6).

Add a most specific slot to g without generalizing it:

This is expressed by the following condition:

$$s \in S_S(g) \wedge f_s(g,s) \geq 0.4 \wedge \forall s' \in super(s) \cap S_G(g) : f_s(g,s') < c(g,s') \tag{2}$$

where $c(g,s)$ has been defined above. The universally quantified conjunction ensures that condition (2) is only fulfilled when (1) is not satisfied. The set $S_S(g)$ is the set of the most specific slots in $\tilde{S}(g)$:

$$S_S(g) = \left\{s \mid s \in \tilde{S}(g) \wedge \neg\exists s' \in \tilde{S}(g) : s' \in sub(s)\right\}$$

Conditions (1) and (2) exclude each other. Fig.10 gives two examples of generalizations constructed according to the conditions above. In both cases $S_S(g)$ = {'disk', 'RAM'} and $S_G(g)$ = {'x-memory', 'memory'} (for $g \in$ {'g1', 'g2'}).

The generalization according to slot entries is performed in the same manner as shown above for slots.

As an important side effect, the *filtering* achieved by discarding slots or slot entries with frequency values too small causes the suppression of those properties at a generalized concept which have erroneously become part of a (subordinate) concept representation. As we have

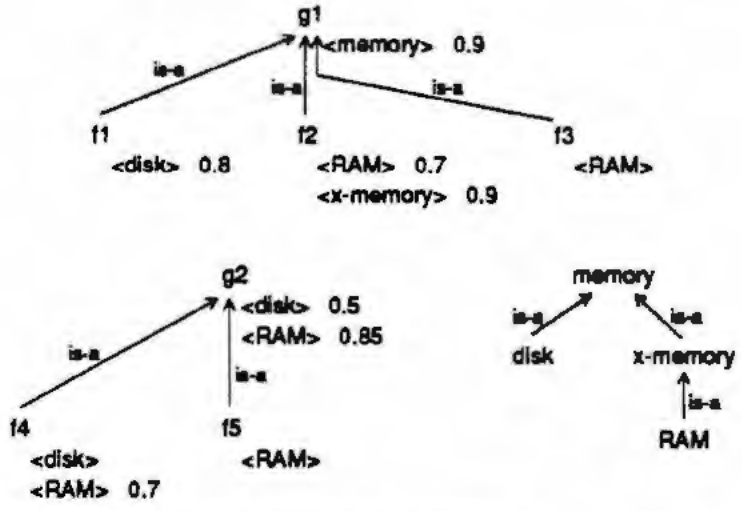


Figure 10: Two examples to illustrate the generalization algorithm

already outlined, such incorrect concept representations may occur because text parsing is partial. Without the filtering resulting from the generalization process it seems to be impossible to realize an automatic knowledge acquisition from texts. This does not only hold for the wit system but is claimed true in general because text parsers with full coverage and total correctness are unrealistic anyway (cf. Sec.2.1).

4. Conclusions

We summarize the most characteristic features of the wit system:

- wit is designed to start its operation with knowledge of only a few, rather generic concepts of a domain of discourse.
- wit realizes an incremental learning and copes with incomplete and uncertain knowledge, as it is necessary when acquiring knowledge from texts.
- Inductive generalization is enhanced by making use of knowledge about the semantics of the underlying representation language.
- The knowledge acquired from a text as well as the knowledge generated by the generalization component is immediately used for further text analysis and further learning (closed-loop learning).
- wit realizes a bootstrapping approach, due to which it needs only knowledge about some very generic concepts of the domain of discourse in advance. This previous knowledge serves focusing the attention of the system.
- wit is able to detect that some words in an input text are names of concepts of the domain of discourse it has not known of before. Getting knowledge of the existence of a concept is the first necessary step in acquiring a representation of it.

- Erroneous parsing results (which are due to the partial parsing) get eventually omitted because many texts are analyzed. Moreover, since the generalization process filters out occasional properties erroneous results do not propagate to generic concept descriptions.

The system is currently being implemented in Prolog on a Sun-4.

Acknowledgements

I would like to thank Klaus Mußgnug for his valuable comments on earlier versions of this paper and for the many discussions.

References

Antonacci, F., Russo, M., Pazienza, M.T., and Velardi, P. (1989). A System for Text Analysis and Lexical Knowledge Acquisition, *Data & Knowledge Engineering*, Vol. 4, pp. 1–20.

Buchanan, B.G., and Mitchell, T.M. (1978). Model-Directed Learning of Production Rules, in D.A. Waterman, and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*. New York: Academic Press, pp. 297–312.

Hahn, U. (1989). Making Understanders Out of Parsers. Semantically Driven Parsing as a Key Concept for Realistic Text Understanding Applications, *International Journal of Intelligent Systems*, Vol. 4 (Special Issue on Knowledge-Based Information Retrieval Systems).

Kodratoff, Y., and Ganascia, J.-G. (1986). Improving the Generalization Step in Learning, in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning. An Artificial Intelligence Approach. Vol.2*, Los Altos: Morgan Kaufmann, pp. 215–244.

Lebowitz, M. (1983). Generalization from Natural Language Text, *Cognitive Science*, Vol. 7, pp. 1–40.

Lebowitz, M. (1986). Concept Learning in a Rich Input Domain: Generalization-Based Memory, in R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Eds.), *Machine Learning. An Artificial Intelligence Approach. Vol.2*, Los Altos: Morgan Kaufmann, pp. 193–214.

Lebowitz, M. (1987). Experiments with Incremental Concept Formation: UNIMEM, *Machine Learning*, Vol. 2, pp. 103–138.

Lebowitz, M. (1988). The Use of Memory in Text Processing, *Communications of the ACM*, Vol. 31, No. 12, pp. 1483–1502.

Michalski, R.S. (1983). 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. Vol.1*, Berlin: Springer-Verlag, pp. 83–134.

Michalski, R.S., and Chilausky, R.L. (1980). Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis, *International Journal of Policy Analysis and Information Systems*, Vol. 4, pp. 125–160.

Möller, J.-U. (1988). Knowledge Acquisition from Texts, in J. Boose, B. Gaines, and M. Linster (Eds.), *Proceedings of the European Knowledge Acquisition Workshop (EKAW'88)*, pp. 25-1–25-16.

Morik, K. (1989). Sloppy Modeling, in K. Morik (Ed.), *Knowledge Representation and Organization in Machine Learning*, Berlin: Springer, pp. 107–134.

Morik, K., and Kietz, J.-U. (1989). A Bootstrapping Approach to Conceptual Clustering, in *Proceedings of the Sixth International Workshop on Machine Learning*, San Mateo/Cal.: Morgan Kaufmann, pp. 503–504.

Musen, M.A. (1989). Conceptual Models of Interactive Knowledge Acquisition Tools, *Knowledge Acquisition*, Vol. 1, pp. 73–88.

Petrie, Jr., C.J. (1985). Extraction of Expert System Rules from Text, MCC Technical Report No. AI-017-85.

Reimer, U. (1986). A System-Controlled Multi-Type Specialization Hierarchy, in L. Kerschberg (Ed.), *Expert Database Systems. Proceedings from the First International Workshop*, Menlo Park: Benjamin/Cummings, pp. 173–187.

Reimer, U., and Hahn, U. (1988). Knowledge-Based Text Analysis in Office Environments: The Text Condensation System TOPIC, in W. Lamersdorf (Ed.), *Office Knowledge: Representation, Management, and Utilization*, Amsterdam: North-Holland, pp. 197–215.

Reimer, U., and Mußgnug, K. (1990). Employing the Semantics of Representation Constructs as Background Knowledge to Enhance Inductive Generalization, Technical Report, University of Konstanz, 1990.

Riesbeck, C.K. (1981). Failure-Driven Reminding for Incremental Learning, in *Proceedings of the Seventh International Joint Conference on Artificial intelligence*, pp. 115–120.

Riesbeck, C.K. (1982). Realistic Language Comprehension, in W.G. Lehnert, and M.H. Ringle (Eds.), *Strategies for Natural Language Processing*, Hillsdale: L. Erlbaum, pp. 37–54.

Slator, B.M. (1989). Extracting Lexical Knowledge from Dictionary Text, *Knowledge Acquisition*, Vol. 1, pp. 89–112.

Sonnenberger, G. (1990). Automatic Acquisition of Terminological Knowledge from Texts as a Test Bed for a More Realistic Text Understanding, Technical Report, University of Konstanz, 1990.

Wrobel, S. (1989). Demand-Driven Concept Formation, in K. Morik (Ed.), *Knowledge Representation and Organization in Machine Learning*, Berlin: Springer-Verlag, pp. 289–319.

Zernik, U., and Dyer, M.G. (1987). The Self-Extending Phrasal Lexicon, *Computational Linguistics*, Vol. 13, Nos. 3–4, pp. 308–327.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Transferring Knowledge from Active Expert to End-User Environment

Kristian Sandahl
Dept. of Comp. and Info. Sc.
Linköping University
S-581 83 Linköping, Sweden
E:mail: krs@ida.liu.se

Anders Brogren
Volvo Data, Dept. 2630 DLI
S-405 08 Göteborg, Sweden

Lars Reshagen
Dept. of Comp. and Info. Sc.
Linköping University
S-581 83 Linköping, Sweden

Abstract

The approach towards knowledge transfer proposed in this paper is called the *active expert approach*. Briefly this approach implies that the expert should take as active part as possible in the creation of the knowledge base. The knowledge engineer should act more like a teacher of knowledge structuring, tool designer and as a catalyst in the dialogue between the expert and the end-users. By doing so, much of the well-known problems with inter-human conflicts, knowledge engineer filtering, expert and user acceptance and maintenance could be avoided. The foundations for our argumentation consist mainly of three different projects, where the approach has been followed strictly. These case-studies are reviewed in the paper together with a nine-phase description of the active expert approach.

1 Introduction

The approach towards knowledge transfer [1] proposed in this paper is called the *active expert approach*. (Sandahl, 1987; Eriksson, 1989) Briefly this approach implies that the expert should take as active part as possible in the creation of the knowledge base. The knowledge engineer should act more like a teacher of knowledge structuring, tool designer and as a catalyst in the dialogue between the expert and the end-users. By doing so, much of the well-known problems with inter-human conflicts, knowledge engineer filtering, expert and user acceptance and maintenance could be avoided.

The basic assumptions behind the active expert approach are that cooperative experts are available and that there is a need for creation of several instances of systems, either as different versions of the same system or as several systems belonging to the same generic problem. The last assumption is necessary for the economic feasibility of this approach, since relatively high initial costs are expected.

Two useful approaches that can be employed in the realization of the active expert approach are the *two-phase development approach*, where generic and background knowledge are first acquired in order to support later stages of knowledge transfer, and the *two-environment approach* (Sandahl, 1987), where the knowledge-base is developed within a highly supportive development environment and later migrated in its delivery environment.

1.1 Background

The "conventional" and most generally applicable technique of transferring knowledge is based on *interviewing approaches*, this means that the knowledge engineer interviews the expert, encodes

[1] We prefer the term knowledge transfer to denote the entire process of carrying knowledge from an expert via a knowledge engineer and a system through an interface to an end-user

his understanding of the knowledge into a knowledge base and later demonstrates a prototype to the expert. We have found several difficulties with this approach:

- Misunderstandings and expert bias decreases the quality of the knowledge base.
- The knowledge engineer becomes an unnecessary intermediary between the expert and the end-users, which potentially could ruin end-user acceptance.
- The costs for knowledge engineering are high.
- The knowledge engineering costs remain high during the entire maintenance phase of the product.
- The often expensive experts have to schedule fairly long meetings. The frequency of these meetings is often high in the initial project planning stages, which decreases the expert commitment.
- It is hard to control the projects. Much of the success relies on the outcome of the personal relationship between the knowledge engineer and the expert.

The value of utilizing knowledge directly originating from domain experts has been recognized by several researchers such as method developers advocating, for example, *protocol analysis* (Fox et al., 1985; Meyer and Peaslee, 1989) or designers of *knowledge acquisition tools* (Boose, 1989) admitting direct interaction with the expert. By collecting the experience being reported in the research community and by running real-life projects, we have gradually refined our approach towards knowledge transfer to the current version of the active expert approach. Our primary goals have been to find means for increasing the probability of end-user acceptance and making the management of expert system development projects safer and less complicated. Similar results have been obtained by other researchers with other methods, for example Linster's *model-guided knowledge acquisition* (Linster, 1989) derived through experience in the field of knowledge acquisition tool design.

1.2 This paper

This paper has two main contributions:

- A description of the active expert approach, and
- a review of our empirical results supporting the active expert approach.

The description of the active expert approach is further elaborated in terms of generally expected effects and practical methodology in the next section. The foundations for the approach consist of three different case-studies. These will be reported in the following three sections: Section three consists of a review of the A^3 project where the method of training of the expert in rule formulation appeared as a most natural solution to the problem. The two-environment approach will also be elaborated. The topic for section four is three projects performed in the same domain, but with different approaches, the first one was knowledge engineer driven, the second was expert driven and the third project was user driven. The effects of active users as well as the two-phase method will be described. Section five describes the development methodology of a project, where the active expert method was used to cope with the problem of involving multiple experts and the problem of knowledge base maintenance for the generic task of fault diagnosis. Some individual systems of this class are currently in practical use. Section six is a discussion of how the active expert approach have affected these three case-studies. Some drawbacks and difficulties with the approach will also be related as a challenge for future research. The entire report is summarized together with the main conclusions in section seven.

2 The Active Expert Approach

In this section, the active expert approach will be further outlined. First, a procedure of typical activities involved in knowledge transfer by active experts will be described. Second, some motivations for the employment of the approach together with a list of expected effects is given. Prerequisites and potential drawbacks of the active expert approach are postponed to the discussion section.

2.1 Procedural description

A knowledge transfer operation using the active expert approach can be divided into nine subactions:

1. **Analysis.** Besides the more conventional analysis work, such as project feasibility studies and requirement specification, the initial phase should also include the acquisition of the meta-knowledge necessary for the project. Examples of such knowledge are: an outline of the reasoning process, the knowledge representation schema needed, the volume of an assumed knowledge-base and the need of different features, for instance, back-tracking. The analysis phase also includes the identification of the knowledge sources. To a large extent, the analysis phase coincides with *knowledge-level analysis* as referred to by other authors (Newell, 1982; Musen, 1989)

2. **Model framework creation.** Based on the experience from the analysis phase, it should be possible to create the model in which the expert will formulate his knowledge. Such a model consists of: a knowledge representation suitable for solving the problem, an interpreter with case invariant problem solving strategies and a language in which the expert enters the knowledge. It should be decided which parts of the knowledge can be coded by the expert. Appropriate transformations from the knowledge input language to the knowledge representation should be determined. Eriksson has discussed different approaches for such transformations (Eriksson, 1989).

3. **Expert Contracting.** At some stage during model creation, the education of the expert in knowledge formalization and the input language begins. Naturally, certain steps to modify the model according to expert's wishes have to be taken. Thereafter, the expert is contracted to enter all knowledge needed in the defined language. If no expert dares to commit himself to the task, then the model has to be further refined.

4. **Environment development.** If some kind of computer-supported input is considered, an environment supporting the expert has to be created. The degree of support could range from simple syntactic checking of entered knowledge to a complete programming environment for experts.

5. **Knowledge entering.** Now, the expert is left to formalize and enter his knowledge in the earlier developed language with only a small amount of knowledge engineer assistance. Naturally, this phase of the project requires a large amount of expert time and should thus be carefully scheduled. In order to work satisfactory, two things are important: First, the expert should be given time to elaborate his knowledge without disturbance or comments. Secondly, the expert should be given some kind of rapid feed-back giving information about how the input might affect the user dialogue.

6. **Exception handling.** It is not realistic to believe that all interesting aspects of a real-life problem could be expressed in the input language or that the expert is capable of fully utilizing the tools given. He might fail to find a way to formalize his knowledge or he might not have been given a language with appropriate expressional power. Consequently, the project should be planned for revisions of the knowledge base by the knowledge engineer.

This could be done either by using the input language in an atypical way, extending the input language or adding a module in the target language.

7. **Fielding.** Since the demands on the knowledge acquisition environment and the final delivery environment probably differ, some steps to translate and/or compile the derived knowledge base have to be taken. In addition to this, the knowledge base will also need to be integrated with other software, for instance an information retrieval system. For further support along this line see (Sandahl, 1987).

8. **Validation.** The validation of software is generally a hard problem. The advantage with the expert system technology is that it is possible to perform several parts of the validation continuously. With the active expert approach, further issues are possible to validate early in the project, for instance the knowledge representation model. A closer involvement of the expert will also contribute to a more complete validation. However, the problem of validating the system against reality still remains.

9. **Maintenance.** The maintenance of the system is in most aspects similar to development. With the active expert approach, much of the responsibility could be given to the expert, provided that appropriate routines are developed. The responsibility of the knowledge engineer is limited to exception handling and larger reorganizations.

2.2 Motivations and expected effects

In the following sub-sections, expected effects from the active expert approach are given. Attention has only been given to features that either are considered to be hard problems where the active expert approach gives better results than more conventional methods, or are unique to the active expert approach.

2.2.1 Improved knowledge base quality

Since some parts of the knowledge base will be provided by a non-programmer, high requirements are put on the ability of the knowledge representation to correctly separate the structure of the knowledge base from the representation of the domain knowledge. Thus, the active expert approach forces the knowledge engineer to spend more time developing a much "cleaner" and more natural representation. Still it will be hard for the expert to judge whether the structure is correct or not, but he will be able to guarantee the correctness of the contents of the knowledge base. This is simply due to the fact the he is the single author of this part.

This immediate verification of the correctness of the entered knowledge is one example of how the active expert approach avoids negative knowledge engineering filtering effects. Other examples of such effects are: unnatural size of knowledge chunks, unclear terminology and human bias.

If the expert is provided with rapid feed-back, he will be given opportunities to explore different ways of coding his knowledge. This will train the expert to gradually become more efficient in knowledge coding, it will eliminate some of the need of routine feed-back from the knowledge engineer, and finally the knowledge base will be based on experimental experience rather than safe academic solutions.

2.2.2 The relation expert - knowledge engineer

Several authors have described the potential conflicts in the interactions between the expert and the knowledge engineer see for example (Hart, 1986) . Some of these could be eliminated or reduced by the active expert approach:

- Boose (1984), and others, argue that the expert feels much more comfortable if he is allowed to elaborate his knowledge alone without being observed.

- The expert becomes less dependent on the knowledge engineer, and gets more responsibility for the project. This *domain expert ownership* (Irgon et al., 1990), leads to increased motivation, which in turn implies increased production.
- The transfer of knowledge is double-directed. The knowledge engineer has to educate the expert in knowledge formalization and possibly the handling of a development environment. Thus, the expert will gain more concrete skills in return for his knowledge than just the possibility to get deeper insights in his domain.

2.2.3 Goal fulfillment

The negative effects of a knowledge engineering filter are doubled in a situation where the knowledge engineer works as an intermediary between the end-user and the expert. If the expert is given a substantial part of the design responsibility, the end-user can communicate directly with him. As a consequence of this, the chance that the "right" system is developed increases, since the end-user can use the correct terminology and refer to old cases to an extent which is impossible for the external knowledge engineer.

2.2.4 The project

It is hard to show that the cost for a single medium-size system will decrease with the active expert approach. This might be true if the knowledge engineer service is brought outside the organization, the development of the tool is not too complicated and the expert is relieved from documentation work during knowledge entering. However, the main idea is that it should be possible to reuse the input language and the developed tool to produce several systems belonging to the same generic application. Thus the average cost for each system will decrease.

A variant of reusing the tool is to allow several experts to use the tool to create individual modules, which are later integrated to a single system. This might also contribute to lower the costs since some parts could be left for less experienced and lower paid experts.

A decrease in calendar time spent per project will also be expected since the expert's time will be used more effectively and fewer meetings have to be scheduled.

A project employing the active expert approach will be safer to run, due mainly to two reasons: First, fundamental issues such as knowledge modelling will be handled early in the project with early involvement of the expert. Second, the moment the expert is contracted introduces a well-defined and important mile-stone in the project. The passing time of this mile-stone gives clear information about the status of the project.

As the maintenance responsibility of the system is given to the expert, smoother integration of the system into the organization can be expected. The main reason for this is that smaller updates are feasible since no briefing of the knowledge engineer for each update is necessary.

Last, but not least, the active expert approach can be a way of handling secrecy matters. If the knowledge engineer is hired as an external consultant, the expert may use the tool to include knowledge that can not be trusted to a knowledge engineer.

3 The Two-Environment Approach: The A^3 Case

This section describes the A^3 case study, a project undertaken in 1983 with an application taken from laboratory medicine (Sandahl et al., 1985). The case illustrates how the active expert approach can be applied with little resources, still yielding a knowledge base of high quality. The necessity and implementation of a two-environment approach will also be related.

3.1 The problem

At the Blood Center of the Regional Hospital in Linköping a MUMPS data base system is running on a VAX 11/750 computer for the management of blood donors and their donations. The

laboratory staff, totally consisting of about 50 persons with laboratory technicians in the majority, is daily investigating about 200-250 blood samples, primarily for AB0-Rh determination and antibody analysis.

The task for the A^3 system (Antibody Analysis Advisor) is to guide a laboratory technician in the first step of a series of tests leading to the identification of suspected irregular antibodies. A correct, though not superfluous, list of different analysis techniques has to be chosen.

Due to the relatively short time devoted to knowledge acquisition (1 man-month), it was almost impossible to understand and implement a deep model of the problem. Hence, a purely heuristic model had to be built. Another effect of the limited resources was the necessity to keep the domain limited throughout the project, even though several closely related areas could be imagined.

Another problem was the difference in requirements between the development environment and the delivery environment. The delivery environment is described above. The development environment had to be more flexible and support incremental development of the system. Among the few expert system shells that were available at that time, EMYCIN running on a DEC-2060 was chosen.

3.2 Knowledge transfer method

The knowledge transfer method used in the project followed the active expert approach outlined in section 2. First, a control structure, to be added to the regime run by EMYCIN, was to be developed by interviewing the expert and rapid prototyping for expert verification.

Second, the expert should be taught the control structure and be allowed to express the rules suited for that control structure. Later, the knowledge engineer would demonstrate a prototype with the most recently included knowledge [2]. The strategy was to start with a small effort of expert training in order to avoid expert bias.

Third, a rule interpreter should be implemented in the MUMPS environment, and the EMYCIN rules should be automatically transformed to the MUMPS data base structure.

3.3 Results

After a short briefing, the double-directed knowledge transfer was started by giving the knowledge engineer some literature in the subject and by assigning the expert to a graduate course in AI in medicine. From this work, the project was defined, the deep model implementation was abandoned and a preliminary control structure was built based on the backward-chaining regime provided by EMYCIN.

Before the first attempt to acquire knowledge directly from the expert began, he was introduced to the concept of rules and asked to break down his knowledge in a memo in natural language trying to think in rules of MYCIN size. The memo defined almost all terminology and all variables that had to be used in the system, but the reasoning knowledge was expressed in a forward-chaining style and showed so many gaps that a prototype system was decided infeasible.

Instead, a small system, strictly adapted to the preliminary control structure, was implemented. The rules were fictitious, but written in the domain terminology. The system was demonstrated to the expert, with an emphasis on the control structure, not the contents of the rules. A few days later, the expert had finished a PM of about fifty rules written in natural language which fitted perfectly for the decided control structure. Besides giving these rules, the PM also ruled out the necessity of certainty factors and negative rules, that is rules which decide that a certain action should not be taken. Hence, the control structure became simplified and, at the same time, verified.

Following the formulation of complementary rules, the expert decided to include a related subproblem into the system. This module was entirely designed by the expert, who found it natural to

[2] The major reason against direct entering rules into EMYCIN was that skills in the INTERLISP editor are required to correct erroneous input.

Full agreement A^3, technician, expert			42 cases
Agreement A^3, expert. Disagreement with technician			34 cases
The technician selects:			
correct but too few techniques	14	cases	
correct but too many techniques	18	cases	
incorrect techniques	2	cases	
Agreement technician, expert. Disagreement with A^3			4 cases
The A^3 system selects:			
correct but too many techniques	4	cases	
No full agreement technician, expert, A^3			11 cases
The A^3 system selects:			
correct but too few techniques	3	cases	
correct but too many techniques	8	cases	
incorrect techniques	0	cases	
The technician selects:			
correct but too few techniques	5	cases	
correct but too many techniques	3	cases	
incorrect techniques	3	cases	

Table 1: *the comparison between A^3, technician and expert*

express himself in a decision-tree, which was straight-forward to transform into backward-chaining rules.

When the knowledge base reached a stable state, it was automatically transformed to the MUMPS format, installed and run under the specially designed MUMPS interpreter.

The knowledge base had been continuously evaluated and reached a proficient status in a 114 rules version. It also seemed to be well focussed. It is, for instance, possible to get no answer at all when providing arbitrary input, which has never happened when real cases have been run. Some knowledge concerning the limits of the system's capability has also been explicitly added.

The migrated version has been tested by end-user representatives. A comparison between the A^3 system, ordinary staff (technician), and human expert (by definition correct) has been made on 91 cases. See Table 1. From this validation, it could be observed that the A^3 system and the expert fully agree in 84 % of all cases and that the recommendations from the A^3 system would never have missed any antibodies present in the sample.

3.4 Conclusions

The project was carried out with the state-of-the-art technology in 1983. Although some advancements have been made since then, much of the results are still valid. If the project were repeated today, a more flexible environment than the EMYCIN system would have been used. Owing to the nature of the problem, the capability of the expert and the high proportion of heuristic knowledge, a system such as AQUINAS (Boose and Bradshaw, 1987) would have been suitable to automate the knowledge acquisition part of the problem.

The problem of integrating the knowledge base into the MUMPS environment still exists. An alternative to implementing a rule interpreter in MUMPS would have been to compile the knowledge base into a run-time version, install that environment on the target machine and use specially designed call-in routines to access the data base. The selection of strategy is still an open question.

Regardless of the methods used, the project supports two assertions:

1. It is feasible to acquire heuristic knowledge directly from an expert, without passing all knowledge through the knowledge engineer.

2. The approach to migrate the system between two environments is feasible and provides a convenient way of completing the knowledge transfer.

The project team was never uncomfortable with the active expert approach, on the contrary, it was believed to be the most natural way to solve the problems. Apart from the problem specification, the fulfillment of the important prerequisite of access to a committed expert influenced the decision in favor of the active expert approach. In addition to this, it was possible to maintain a high frequency of short meetings during the most intensive phase of knowledge entering.

However, during one formulation-input-verification cycle, a quite large amount of knowledge was entered at the same time. As a result of this, the expert needed some help to convince himself that the combination of all individual knowledge chunks behaved correctly. This observation supports the intuitively appealing strategy to strive for inclusion of very small amounts of knowledge in several frequent cycles.

Otherwise, the project benefitted from the fact that the expert really knew what the system knew. This made the evaluation very efficient, since errors due to lack of knowledge could easily be recognized and taken care of. It also helped in internal marketing and problem restriction.

3.4.1 Practical hints

A few practical details contributed substantially the successful transfer of the knowledge. They are briefly related here for the reader who wants to try a similar method.

First of all, the knowledge engineer needs to be very careful in the education of the expert. If the expert is forced into an unnatural knowledge representation form early, he might exclude important information due to his inability to express it correctly. It is better to give the expert too little information in the beginning and complement it later as was done in the A^3 case. Even though the derived information is not immediately suited for inclusion in the knowledge base, expert formalized knowledge is seldom entirely useless. A detail related to this matter is the use of prototype system for education. The small system mentioned above contained related information, but much care was taken to make it totally irrelevant in order to avoid the expert copying the rules; the goal was to teach him only the control structure.

Not all knowledge is possible to obtain directly from the expert. This knowledge should be kept separated from other kinds of knowledge for the same reason as the separation of domain knowledge and control knowledge.

The first quick implementation of the MUMPS coded inference engine gave performance problems. An early consultation lasted about ten minutes, compared to two to three minutes in the EMYCIN environment. Further optimizations brought the time down to EMYCIN performance. This calls for warning against too naive implementations of delivery systems in specialized environments, but does not discourage more serious attempts. Naturally the development of hardware performance removes some problems. However, this might not help for larger knowledge bases. Compared to the requirements for decision support, A^3 could only claim to be considered as a sub-system of about a tenth of the size of a regularly working system.

4 The Active User: The Social Insurance Case

Three different systems were developed in the domain of social insurance during 1986 and 1987. The third of these, developed with the active expert approach, is the main topic of this section, which will describe how active users contributed to the fulfillment of project objectives.

4.1 Domain background

Swedish social insurance contains several types of benefit for parents. The most important are:

1. Parent's allowance in connection to the birth of a child (FP). Payed leave of absence is given to parents who refrain from work while caring for a small child.

2. Parent's allowance for temporary child care. (TFP). Parents are compensated if they have to refrain from economic activity to take care of sick children.

These benefits are administered by regional social insurance offices, and the volume of FP and TFP matters from September 1984 to September 1985 were 112,277 and 2,023,528 respectively. The data processing is centralized and kept by the supervisory authority.

The parent's allowance cases are processed by handling officers who are specialized in health insurance. They have to decide whether the basic prerequisites for the allowance are fulfilled and determine the size of compensation. In connection to these tasks, the handling officer has to:

- handle exceptions, such as late applications, very young parents and adoption.
- check and judge the correctness of the information given by the applicant.
- retrieve and apply relevant knowledge from sources such as laws, official working routines, court decisions and interim regulations.
- assist the applicant in making correct and optimal claims.

The current policy to decentralize both the computer processing and the responsibility, and to turn the handling officers into generalists, calls for a higher degree of locally available advanced decision support tools.

4.2 The first two systems

In the following two subsections, the two earlier experiments will be reviewed in order to give the appropriate background to use the active expert approach in the third major system under review.

4.2.1 The TENOR system

The goal for the first project, called TENOR (Swedish acronym), was to demonstrate how parts of social legislation could be formalized, and how such a formalization would behave if it was implemented in a system. The domain was general processing of TFP cases with an emphasis on judging whether the applicant's claim to refrain from economic activity was relevant.

The implementation was mainly technique driven and the amount of expert time was small, only two meetings. Consequently, much of the knowledge in the system was derived from written sources, and it was up to the knowledge engineer to decide whether to include certain knowledge or not. Not surprisingly, the system behaved logically accurate, but did not correctly focus on relevant issues. For example, extremely rare situations were taken care of in the very beginning of the dialogue, and matters concerning the employer of the applicant were emphasized more than social insurance. These are typical effects of low expert involvement.

The main contributions from the project were:

- The demonstration of the feasibility of the technique to formalize legal knowledge was successful.
- The knowledge engineer correctly identified the need of combining expert system and information retrieval techniques in this area.

4.2.2 The ALTO system

Following the feasibility study was the ALTO (Swedish acronym) system which was somewhat more ambitious: To develop a complete system in a limited area which could reach expert acceptance. Since the steering committee had started another more user-oriented project in parallel to the ALTO project together with a competitor to us, no user involvement was judged necessary.

The work was carried out with two experts from different authorities, who were interviewed in a conventional way. One successful active expert-oriented method was the use of decision tables

which were constructed by the experts and the knowledge engineers together. The acquisition of knowledge was split into two sub-activities: to refine the TENOR knowledge base, and to introduce an area for application of deeper reasoning, handling of claims that had reached the insurance office too late. Within the second activity, a method where the user could browse a library of old cases before answering the questions was developed.

The parallel project was never completed, and in order to give the users intended something concrete, the ALTO project was demonstrated instead. Owing to unknown reasons, the users' trade union representatives never compared the system to its original goals and forced the steering committee to focus on other things rather than a serious evaluation of the project. However, subjective statements from the experts and a post-graduate law student, who had been observing the knowledge acquisition process, were available. Several of these seem to favor development more oriented towards active experts:

1. The interview form is badly suited for legal applications. Things are often complicated, and thorough investigations are often needed in order to answer yes or no to a question. The expert might be better off if he were allowed to make frequent but small updates of the knowledge base in a phase where the main activity consists of investigations by the expert himself.

2. The need of working alone is also expressed by the expert concerning the testing of the program.

3. Even though not experienced in the ALTO project, the experts report a potential danger that the expert might identify himself too much with the knowledge engineer, if he is always present, and incorrectly look for quick and too exact solutions.

4. In legal applications, the linguistic formulation of texts and questions is extremely important, and should be done by people who are educated in law. In the project, a knowledge engineer did acceptably well, but the final formulation had to be obtained directly from the experts any way.

5. The legal status of systems similar to ALTO is as yet unclear. The selection of the knowledge to be included in the system has to be made with great legal knowledge in order to assign the system a correct status in order to, amongst other things, properly handle responsible matters.

4.3 The problem

The domain problem for the third system, called SOPRANO (yet another Swedish acronym), was to cover the more difficult area of FP. The main feature of this domain is that a system giving decision support has to tightly integrate reasoning with information retrieval. The information is spread over several differently accessible sources, which complicates processing using manual routines. The goal was set to create a system which could be practically validated by several users at different levels.

Several more or less well-founded requirements were put on the development process:

- The development style had to differ substantially from the ALTO system due to political reasons.
- The trade union demanded a high degree of influence over the development work.
- All knowledge entered should be tagged with a first-hand written source.
- High attention should be paid on legally correct formulated texts.
- The system should strictly focus on the needs of the users. It should, for instance, not be possible to handle a complete case without any prior background knowledge.

- The knowledge should be accessible in a flexible manner.

Expertise was generously provided. The experts were selected from the highest level where individual cases are processed.

4.4 Knowledge transfer method

The active expert approach was taken with a high degree of user involvement. The trade union accepted a solution where the entire analysis phase would be spent together with two handling officers working at a local branch. The surface structure of the interface and the scope of knowledge should be derived. The users should actively assist in interface design.

Later, the experts should be contracted and asked to break down the ordered knowledge into chunks consisting of a text in natural language and an encoding of the text, which should be used by search algorithms and by the knowledge engineers in the creation of the rule base. If necessary, negotiations should be started directly between users and experts to ensure that correct and relevant knowledge would be included. After this step, the users would couple the knowledge chunks to the interface.

The EPITOOL/DE environment version 1.3 was utilized, mainly due to its flexibility, since it contains a procedural programming language, but also for its features for compilation of the source code for migration into a delivery environment that were planned for version 2. This was successfully demonstrated with version 2, released during the end of the SOPRANO project.

4.5 Results

The outcome of the analysis phase with the end-users resulted in a system centered around an inference tree of the domain. The tree should be explicitly shown on the screen, and by selecting a node at an arbitrary level of abstraction, the user could follow two main strategies:

1. He could use the system as an information retrieval system by selecting among a variety of search conditions. Either information is sought by joining information from specified nodes, or the entire knowledge base is searched for potentially relevant information

2. The user could use the inference engine. The system then starts to solve the problem related to the selected node by asking questions about the actual case. After the answer is found, relevant information for the case could be accessed with three different levels of focus. [3]

The users were given the task to design the inference tree, after having watched the knowledge engineer creating a preliminary sub-tree of the problem. All texts and names of the different nodes were provided directly by the users. The domain was decided by the users, against advice from the experts, who advocated a more narrow, but deeper approach. The users founded their decision upon interviews made by themselves at different branches in the region.

This arrangement gave the knowledge engineers time to concentrate on knowledge representation, and a prototype system was created and demonstrated. As the functions of the prototype were accepted, the users ordered knowledge directly from the experts. Only small conflicts appeared. In most of the cases, the experts wanted to provide a bit more knowledge than was actually ordered. Each matter of this kind was settled at conferences without knowledge engineers present.

The experts were taught how to break down their knowledge into chunks. The correct size of the chunks was discussed; the experts argued that some context information might get lost if the size became too small, whereas the knowledge engineer claimed that the matching and inference routines would ensure the right context. An agreement was accomplished, and the work of knowledge splitting was very well performed by the experts.

A knowledge chunk consists of a text in natural language, which is assigned a piece of code consisting of three sets of object-attribute-value triples. The sets of triples are:

[3] Hence it is possible to use the inference engine, not only to solve problems, but also as a tool to focus on the information wanted.

	statement number							
rating	1	2	3	4	5	6	7	8
5	2	9	2	3	1	0	5	3
4	22	22	27	25	15	25	27	12
3	6	2	6	6	11	10	3	8
2	5	2	0	1	6	0	0	10
1	0	0	0	0	0	0	0	2

statement number	statement
1	It is easy to use the system
2	The knowledge in the system is relevant and complete
3	The dialogue when running the inference mechanism is good
4	It is easy to use keywords and other concepts from the domain
5	It is easy to find the correct node for the problem
6	The window technique is nice to work with
7	The language used in the system dialogue is suited to the users
8	The system controls your work

Table 2: *results of a poll amongst different end-users*

primary description contains triples, which, if all of them are evaluated to true in a context, necessarily identifies the chunk as relevant in that context

secondary description contains triples, which, if evaluated to true, give support for the relevance of the chunk

excluding description contains triples, which, if evaluated to true in a certain context, identifies the chunk as not relevant in this context, regardless of the result of the evaluation of the other descriptions

A fully agreed convention of how to code the chunks was never accomplished, but the work was acceptably well performed. The triples were entered at the same time as the texts by the experts. Their environment consisted of a text editor, a sub-set of EPITOOL commands and some elementary checking routines.

The input language was gradually extended to meet the demands from the experts. Mostly, this meant that the vocabulary of objects and attributes was extended to make finer distinctions between knowledge chunks. As the experts became more interested in handling of the computer system and equipment, they were taught more advanced commands. Some routines were discovered by the experts themselves through experiments.

As the users got the chunks, they had to couple them to the interface by assigning them to the nodes that were considered to be relevant. The users also participated in the encoding of the chunks to a limited extent.

Totally 300 chunks, mostly consisting of excerpts from legislation, were created and entered by the participants. After this stage, a rule base to be used by the inference engine in solving the problems related to the nodes in the inference tree was designed by the knowledge engineers. Due to a delay of hardware delivery, this activity was performed on another machine, which further delayed the total integration of the system and optimizations necessary for delivery. As a result of this delay, the delivered system was not satisfactory debugged, but did after all get a positive reception. See table 2 for details. In the table, the results from two different investigations using the same material have been added. A rating of 5 means full agreement with the statement, and 1 means full denial. The statements have been transformed from interrogative form and translated into English by the authors.

The performance was measured by giving 10 people from different branches a set of problems which they should try to solve with the system and log their activities. 88 logs were obtained, 90 %

of them claimed that the session had given the correct answer. The system had been used purely for information retrieval in 34 cases, and the inference engine was utilized in 54 cases. Concerning the interface, 75 consultations easily obtained the advice, whereas in 12 cases, the user found it difficult to access the system. In 56 of the cases, the user chose the right node from the beginning; in 27 cases, several nodes had to be accessed. A preliminary test of the same size performed at the site of development showed similar figures. After these tests, the system has been demonstrated to and evaluated by representatives from all branches in Sweden using specially selected cases.

4.6 Conclusions

The case study has illustrated how the active expert approach can be utilized to provide for a high degree of user participation, and how this arrangement can be used to enhance goal fulfillment of a project with real-life requirements.

Even though some heuristic knowledge was never included and the system was never decently debugged, the performance and reception among the users seem to be most promising for a first prototype. Due to the active expert approach, all effort in the development has been put on the domain, that is the experts have directly tried to provide the users with problem-solving knowledge in their standard terminology. Observe that no negative responses were given to statement number 7 in the table above. As a side-effect of this, computer specialists have argued that the language used in the interface is not understandable, and some are not impressed by a brief look at the functions of the system. It is probable that if more emphasis had been put on creating a demonstration system, less effort would have been spent on giving the user what they really wanted.

The knowledge transfer was completely successful in the sense that the correct knowledge chunks were identified and constructed. This is supported by the high amount agreement with statement 2 in Table 2. Regarding the coding and linking of the chunks in the interface, some confusion appeared. This was due to the low amount of feed-back to the experts and users, see practical hints below. Even though much could be sorted out by the knowledge engineer explaining what would happen if a certain chunk was coded in a certain way, this could not replace a true run of a complete prototype. As an effect of this, the users and experts did not fully trust the inference and search routines to find all relevant chunks. A tendency to partially evaluate an imagined inference engine with respect to the coding was observed, which delayed the work as well as making the focusing mechanisms less specific. The lesson from this part of the project was that rapid feed-back is crucial for development according to the active expert approach.

The inference rules were created by the knowledge engineer. This might seem to obstruct the active expert approach. However, each rule corresponds exactly to a knowledge chunk created by the experts, and the encoding of the chunks were used in writing the rules. Long afterwards, it was subjectively judged that the chunk code could have been automatically transformed to rule code in about half of the cases, if only the experts had divided the triples in a left and right hand side of the rule. Since the creation of rule bases often requires a large amount of tuning, efficient feed-back routines would have been necessary if expert knowledge entrance had been arranged.

4.6.1 Practical Hints

The SOPRANO project would not have been possible to complete without truly committed experts and end-users. This might have been hard to achieve if the TENOR and ALTO projects had not been used as an opportunity to catch the interest of the experts. After those projects, the main expert himself negotiated to be relieved from much of his ordinary duties in order to participate in the project. The advice resulting from this is to wait with the active expert approach until the right amount of expertise can be contracted. Furthermore, the role of a pilot project could be extended from experiment to prepare a way for larger active expert driven efforts.

The choice of people is important. The golden rule is to select people who are working every day with the domain. Another fully reasonable choice could have been to let the trade union representative represent the users, and use highly qualified jurists as experts. Most probably such

an arrangement would have led to an executable system, but the power of goal fulfillment of the active expert approach would not have been utilized.

5 A Generic Problem: Fault Diagnosis

This section illustrates how a more general model with supportive tools could be developed within the active expert framework in order to cope with several practical problems for an entire generic application. Several shortcomings of the SOPRANO project were successfully attended. The project is called FOOD (Functional Object Oriented Diagnostic-environment) and was started at VOLVO, in 1987.

5.1 Domain background

The selected domain was fault diagnosis and management support of technical systems, especially different kinds of engines designed by VOLVO. This kind of problem is supposed to be well-known and has been thoroughly treated elsewhere in the literature, not the least in the area of quantitative physics and deep modelling.

5.2 The problem

The life-cycle of the interesting products is fairly large, typically three decades, and involves the engagement of several specialist experts. In this application, an expert system would not only capture their knowledge, but also serve as a communication medium among them and end-users. To guarantee the consistency of the system, it is important to facilitate for rapid entrance and rapid and transparent feed-back of entered knowledge.

The market calls for several specialized systems or system modules. A realistic case-study of a single product shows a need for about 100 such modules with 10 man-weeks expert time for each of them. If these modules were to be developed in sequence by a single knowledge engineer, the system would be delivered in about 20 years, which, of course, is totally infeasible.

The details about most of the products are, at least during a certain phase, strictly confidential. Thus, it would be desirable if the formulation of the knowledge could be handled within the organization without briefing a consultant knowledge engineer. This requirement puts extra high demands on the transparency of a supportive environment for system development.

5.3 Knowledge transfer method

During the development of a fairly simple model for expert knowledge, an environment allowing the expert to directly input his knowledge was to be gradually developed. The input language chosen was supposed to be automatically transformed into the target language EPITOOL. This work was to be achieved through the design of a pilot project to guarantee a high degree of expert participation.

A highly transparent user interface was to be added for test-runs and delivery as well as routines for generation of hardcopy feed-back of entered knowledge.

To fulfill the secrecy requirements, the knowledge engineer had to be removed after the expert contracting point.

5.4 Results

Since an earlier project with the goal to make a complete list of all possible malfunctions of the engine selected for the pilot project had failed, the focus of the earlier stages of knowledge acquisition was directed towards the development of a functional model.

Before the model development started, ten different functions were selected from written documentation. A test criterion was established, stating that the system should be able to produce the same answer as the manual to gain acceptance of the model.

A double-directed transfer of knowledge was initiated, where highly frequent real cases were utilized. The primary goal, as far as the knowledge engineer was concerned, was to gain a high-level understanding of the cases in order to acquire the expert's view on the problems rather than actually learning fault-finding. The expert was taught how knowledge in this domain could be formalized. In close cooperation, the two agents developed a model, in which the expert was able to further practice his knowledge formulation skills. In early stages, the knowledge was simply entered as text and manually coded into the target language to show the expert the impact of his modifications to the knowledge base. As the model gradually became more acceptable, automatic routines were added until a final environment was achieved.

The knowledge representation could briefly be described as follows: The expert creates an object corresponding to a real or abstract component. Each object has five attributes:

User prompt - a text asking the end-user to describe the status of the object.

Help text - a text clarifying the above question

State list - a list of possible states which the object could be in

State texts - a list of texts where each text corresponds to a state in the state list. These texts are used in the dialogue with the end-user.

Remedy text - a text telling the end-user what to do when repairing the object. This attribute might be empty.

The expert also creates the states, which have four attributes:

Conclusion - a text describing the state

Action - a text telling the end-user what to do if the state has occurred. This attribute might be empty

Condition - a boolean expression with terms on the form *object(state)* meaning that *object* is in *state*. If the condition evaluates to true, the calling object is inferred to be in *state*

Remedy text - a text telling the user what to do when condition is not satisfied and it is known that the calling object is in *state* or the other way round.

Besides editors and checking routines to support the expert's work of entering knowledge in this language, the environment also consists of a user interface with an explicitly shown hypothesis-graph as a dominant part. This graph allows both the expert and end-user to follow the reasoning of the program for checking and educational purposes respectively. Added to the expert's routines are also routines for the production of a highly readable report showing objects and states.

To return to the history of development, a contracting point was reached after 60-70 % of the project time. This point included a commitment of both actors. The expert promised to enter all relevant knowledge, and the knowledge engineer guaranteed that the implemented environment would behave as earlier demonstrated. From this point only small adjustments were shown to be necessary and the expert entered 100 objects with interactions through states on his own.

The creation of the pilot system engaged one knowledge engineer for six man-months and an assistant knowledge engineer for about three man-months. It is currently used for documentation purposes within the pilot branch of VOLVO, and the experts are totally satisfied with the system as an aid for structuring their knowledge. Through the use of the much appreciated hypothesis tree, they have been able to detect and correct several potential bugs.

From the pilot project, a generic environment for fault-diagnosis was derived and successfully tested at two different branches of VOLVO. The methodology used in this project is currently adopted for large-scale purposes at VOLVO.

5.5 Conclusions

The active expert approach proved to be a valuable tool in solving the problems in section 5.2. Maybe it is the only solution to the secrecy problem.

In this project the major shortcomings of the SOPRANO project were successfully taken care of. There was no confusion about the coding of knowledge due the ability of providing the experts with rapid feed-back. Even though the interface was primitive in the start of the project, some kind of feed-back was available during almost the entire development phase.

No performance problem occurred, mostly due to employment of a fairly simple model of knowledge representation. The simplicity also played an important role in selling the product to other experts. The capability of solving the generic problem of fault-diagnosis could be explained by the choice of a non-complicated knowledge representation.

In the pilot project described, no logical consistency checking of the input is performed. In spite of this, the non-programmer experts have been able to keep the consistency using only common sense and their knowledge about the domain. This contradicts the statement that automatic consistency checking always is a critical part of a knowledge acquisition tool. The question of how valuable it is to add often expensive consistency checking routines still remains to be decided.

5.5.1 Practical hints

During the work, the experts hesitated over the vast amount of input required to complete a proficient knowledge base. This was elegantly solved by letting one of the apprentice engineers perform the formalization and entrance of a majority of the knowledge. Through this arrangement, the knowledge acquisition process was used as a most effective tuition method saving the time of the expensive expert originally required for both knowledge entrance and teaching.

6 Discussion

The discussion consists of two parts: the first one enlightens the most distinguished impacts of the active expert approach observed in the three projects reviewed above, whereas the second part formulates, and partly explains, some commonly heard objections against following the approach.

6.1 Impact of the active expert approach

The active expert approach showed to be critical for the success of the projects in the following senses:

- In the A^3 project, the knowledge of the expert was necessary in performing the delicate work of consistently limiting the domain of the system. The small amount of calendar time devoted to the project gave no room neither for experiments nor tuition of the knowledge engineer. The result of the approach became obvious in two ways: first, the explicit identification of problems where a human expert should be called, and second, the addition of the related sub-problem described in section 3.4. Since no errors were made here, time was saved.
- Judging from the testimony given by the expert after the ALTO project in section 4.2.2., the active expert approach seems to be the most desired with respect to the expert's working conditions and the nature of the domain. Following the same source, a correct formulation of texts need to come directly from an expert. Furthermore, the active expert approach became a natural and efficient way of taking care of the valuable direct dialogue established between the experts and the end-users.
- In the FOOD case, the major contribution from the active expert approach was the fulfillment of the secrecy requirements. It is hard to imagine any other approach performing better in this respect. The approach also gave a promising solution of the maintenance problem.

Apart from solving critical problems, the active expert approach emerged as a most natural component of the development method applied in the three projects. In the A^3 case, it would have been wasteful not to utilize the enthusiastic expert for a deep engagement, in the SOPRANO case, it was necessary to offer something new to win user acceptance, and in the FOOD case, the approach was favored in the development of several systems in a generic application.

Another common feature of all the three projects was the invaluable support given by the experts for the internal marketing of expert system technology within their organizations.

It is interesting to notice the difference between the arguments for using the knowledge base improvement properties of the active expert approach in the A^3 and FOOD cases. In the A^3 case, the argument was to rapidly develop a heuristic, not necessarily deep, knowledge base. On the other hand, in the FOOD pilot project, such a shallow model had failed, and the active expert approach was to be used to develop a deep model. At a first glance, this might look as the contradictory property. In fact, it reflects how two different aspects of expertise have been utilized, namely the ability to compile and use compiled knowledge gathered by experience, and the ability to recreate lower order principles and reason with them. Furthermore, this observation illustrates the importance of finding a model which unifies with the goals of the project. Once the expert has been contracted, the active expert approach seem to amplify the goal fulfillment property of the expert system technology in general.

6.2 Potential problems and questions

Loss of knowledge

Expert bias as described in the literature is considered to be a problem in acquiring knowledge (Cleaves, 1987). One example was given in the expert's comments to the ALTO project in section 4.2.2. where experts are over-helpful and go for too quick results to please the knowledge engineer. An argument against the active expert approach could be the very strong bias towards the knowledge representation chosen. This may, for instance, tempt the expert to exclude knowledge, which is hard to formalize, from the knowledge base.

When discussing bias, it is important to distinguish between positive and negative bias with respect to the goals of the project. If the goals were to create as good picture as possible of the internal reasoning of the expert's brain, surely all bias induced by the active expert approach would be negative. However, since the goal is to create an accepted system at reasonable cost, most of the bias from the approach is positive. The experts, especially from the FOOD project, testify that the model used often is of great help in formalising their knowledge, and that they actually gain rather than loose insights in their own domain when working with a knowledge representation model. The fear of exclusion is often exaggerated, in neither of the systems described had shortage of knowledge posed a hard problem; it has rather been the other way round: to decide what to exclude and still create a balanced and consistent knowledge base as was the case in the A^3 system.

Several arguments for simple knowledge representation schemata have been forwarded in this paper. The reasons were mainly time performance, expert acceptance and generality. It is admitted that there is a risk of missing knowledge by application of an over-simplified representation. However, there are two dimensions of representation complexity along which simplicity is favored:

1. If the knowledge representation is too complicated, the expert might not be able to understand it, which could lead to wrongly designed knowledge chunks. This was almost the case in the SOPRANO project where the experts did not fully accept all details of the effects of their coding. Well-designed feed-back routines are of course a way to push the limit of how complex representation the experts might accept.

2. If the knowledge engineer is caught in "the AI trap" and strives too hard to create a maximally natural model of the actual reasoning of the expert, it is likely that good work will receive acceptance by the expert. It is, however, most likely that such a solution will show weak time performance. Such a mentally simple task as non-linear planning is, for instance, NP-hard. This effect was also observed in the early phases of the SOPRANO project.

The creation of a feasible knowledge representation is maybe the hardest task for the knowledge engineer, but the situation is not hopeless. Naturally, good practice with the active expert approach will help future projects and ongoing research strive for development of more advanced knowledge acquisition supporting environments.

Four pieces of practical advice on this matter can be given:

1. Ensure that enough time is given to the analysis phase. About one third of the total project time is not too much.
2. Again, trust the knowledge of the expert. If the expert honestly signs a contract saying that all necessary knowledge could be entered given the derived tools, then there is strong evidence that the model will work for an acceptable large class of cases.
3. Make it possible to perform simple patches in the knowledge representation scheme. In all reported cases, it was possible for the knowledge engineer to manually access a more general tool than was agreed with the expert. However, this was used in very limited extent, and not at all in the FOOD project.
4. Provide for expert and end-user feed-back from the beginning by including some small pilot-project in the analysis work preferably in a simple rapid prototyping environment.

Expensive expert efforts

An inherent property of the active expert approach is large consumption of expert time, which of course adds to the development costs.

The answer is to regard the expert time as an investment which hopefully will pay back in form of:

- improved knowledge base quality,
- smooth solutions to the problems of organising maintenance,
- high quality contribution by the expert in the marketing of the system, and
- expert training due to the double-directed knowledge transfer principle.

Furthermore, in the projects reviewed, the experts have put much personal engagement in the project, and there has never been a problem to persuade them to take a few hours over-time during critical parts of the projects.

The active expert approach gives also an opportunity to combine knowledge transfer with other knowledge management tasks, such as reporting experiences or tuition, as in the FOOD case.

As with all investments, the alternatives should, of course, be seriously examined. A problem might be the difficulty to translate the gains of the active expert approach into monetary units, a subject which deserves more attention by researchers.

Long-term problems

So far, this report has only covered a part of the total life-cycle of an expert system ranging from knowledge transfer of the first prototype and the early phases of maintenance. If the project is driven beyond this point, it is likely that other more long-term problems would appear. Examples of such problems are:

- The difficulty to maintain the original intentions of the knowledge representation schema, which causes different parts of the knowledge base to behave differently depending on the time it was entered into the system.
- Inconsistencies introduced by multiple experts. Two different experts might, for instance, use two different names for the same parameter in the knowledge base.

- The supportive environment could be expected to be in greater need of frequent updates than other similar products since the experts gradually receive feed-back from the end-users.

To simplify the solutions to long-term problems, it would be desirable if future support environments could be used for relatively long periods between reorganizations of the tools and the knowledge base in order to give researchers realistic information about the expected effects of these environments. Furthermore, software support to lower the cost of the reorganizations will most probably be needed.

Almost no experience of the kind related above has been reported, and the authors are currently focusing their research on these questions.

7 Summary and Conclusions

In this paper, the active expert approach, a significant core activity of several of several expert system implementation projects carried out by the authors, has been reviewed. The essence of this approach is that the expert should take as active part as possible in the development of the knowledge base. The underlying assumption behind this idea is that the knowledge of the expert is unique in the sense that it is almost impossible to transfer it correctly and directly to another human a reasonable cost.

A thorough analysis phase and a well-defined expert contracting mile-stone are important ingredients of an active expert driven project. They are not only beneficiary for the result, but also help in the management of the project, since these activities have close counterparts in traditional project management.

In three projects, the impact of the active expert approach has been illustrated and analyzed. Positive effects are: High knowledge base quality, an amplification of the goal-fulfillment properties of expert system technology, deeply engaged experts and end-users and systems maintainable within the application's organization. There are no claims made that the active expert approach is the only way to achieve these goals, but it is remarkable how well goals naturally are fulfilled once the approach taken in the three projects is applied.

From the projects reviewed, it seems as if the active expert approach is independent of application and degree of automation. Currently, it seems most natural to try to identify useful generic applications and strive for a highly automatic supportive environment for development of systems within this application.

Owing to the scarce amount of background material, the results in this paper should be considered as preliminary. The active expert approach is currently undergoing refinements, as more projects are undertaken or otherwise come to the authors' knowledge. The goal is to arrive at something more solid than just an approach, that is an empirically and theoretically well-founded method for expert system design.

Acknowledgments

Kristian Sandahl has formulated the active expert approach and performed the analysis of all work in this paper. He was the chief knowledge engineer in the A^3, Tenor, Alto and Soprano projects.

Anders Brogren worked as knowledge engineer in the Alto and Soprano projects. He was the chief knowledge engineer in the FOOD-project.

Lars Reshagen was responsible for the migration and evaluation of the A^3 system.

Nosrat Shahsavar at the dept. of. medical informatics at Linköping University assisted in the migration of the A^3 system.

Göran Forslund worked as knowledge engineer in the Tenor project.

The authors wish to thank all enthusiastic experts and end-users.

Competent comments on previous drafted versions of this papers has been given by Sture Hägglund, Henrik Eriksson, Peter Eklund and Tomas Sokolnicki at Linköping University, Sweden.

References

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

Boose, J. H. (1989). A survey of knowlegde acquisition techniques and tools. *Knowledge Acquisition*, 1(1):3–37.

Boose, J. H. and Bradshaw, J. M. (1987). Expertise transfer and complex problems: Using aquinas as a knowledge-acquisition workbench for knowledge-based systems. *International Journal for Man-Machine Studies*, 26:3–28.

Cleaves, D. A. (1987). Cognitive biases and corrective techniques: Proposals for improving elicitation procedures for knowledge-based systems. *International Journal for Man-Machine Studies*, 27:155–166.

Eriksson, H. (1989). *A Study in Domain-Oriented Tool Support for Knowledge Acquisition*. Linköping Studies in Science and Technology, Licentiate thesis 181, Linköping University.

Fox, J., Myers, C. D., Greaves, M. F., and Pegram, S. (1985). Knowledge acquisition for expert systems: Experience in leukaemia diagnosis. *Methods of Information in Medicine*, 24(2).

Hart, A. (1986). *Knowledge Acquisition for Expert Systems*. Krogan Page.

Irgon, A., Zolnowski, J., Murray, K. J., and Gersho, M. (1990). Expert system development a retrospective view of five systems. *IEEE Expert*, 5(3):25–40.

Linster, M. (1989). Towards a second generation knowledge acquisition tool. *Knowledge Acquisition*, 1(2):163–183.

Meyer, M. A., Mniszewski, S. M., and Peaslee, A. T. (1989). Using three minimally biasing elicitation techniques for knowledge acquisition. *Knowledge Acquisition*, 1(1):59–71.

Musen, M. A. (1989). Conceptual models of interactive knowledge acquisition tools. *Knowledge Acquisition*, 1(1):73–88.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

Sandahl, K. (1987). *Case Studies in Knowledge Acquisition, Migration and User Acceptance of Expert Systems*. Linköping Studies in Science and Technology, Licentiate thesis 127, Linköping University.

Sandahl, K., Hägglund, S., Hildén, J.-O., Rehmnert, R., and Reshagen, L. (1985). The antibody analysis advisor and its migration into a production environment. In *Proc. of the 1st Int. Expert System Conference*, London.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Integrated Knowledge Acquisition from Text, Previously Solved Cases, and Expert Memories

Franz Schmalhofer, Otto Kühn, Gabriele Schmidt

German Research Center for Artificial Intelligence
University Bldg 57
Erwin-Schrödinger Str.
6750 Kaiserslautern

e-mail: schmalho@informatik.uni-kl.de

Abstract. Within the model-based knowledge engineering framework, an integrated knowledge acquisition method was developed for a complex real-world domain with different traces. By having an expert constructively explain the previously solved cases with more general information from other traces of expertise (text, memories) a model-centered knowledge base is constructed. The proposed method allows for an early knowledge validation where relevance, sufficiency, redundancy, and consistency are already assessed at an informal level. The early knowledge validation efficiently prepares the consecutive knowledge formalization. Through a cognitively adequate model of expertise and the explanation-oriented knowledge elicitation procedures, user friendly second generation expert systems may be developed.

1. INTRODUCTION

Within the last decade, knowledge acquisition has emerged as an important area in expert system development. Whereas some researchers investigated the elicitation of knowledge independent of a sophisticated model of expertise (Boose & Bradshaw; 1987; Diederich, Linster, Ruhmann & Uthmann,1987), others have emphasized the importance of developing appropriate models of expertise (Clancey, 1985; Chandrasekaran, 1986, Breuker & Wielinga, 1989). Obviously, a complete knowledge acquisition technique must include both, the formation of a model of expertise, which can be implemented on some computer system as well as the the elicitation of corresponding knowledge.

Through the knowledge elicitation research various methods for interviewing humans and probing their expertise have been imported from the behavioral sciences (Hoffman, 1989) and their usefulness for the development of expert systems was investigated. The development and application of respective tools has shown how repertory grids, think-aloud methods, scaling techniques, or having the expert perform special tasks (and other data collection and analysis procedures) can be utilized for knowledge acquisition.

Research on model-based knowledge engineering has produced a number of principles and a systematicity on how models of expertise can be formed or selected from a library of models. Problem classes (Clancey, 1985), generic tasks (Chandrasekaran, 1986) and interpretation models (Breuker & Wielinga, 1989) are somewhat different but similar incarnations of models of expertise. A good model for a desired application can be elaborated in empirical and theoretical analyses.

In order to develop a successful expert system for a complex real world application, such as computer aided planning, model-based knowledge engineering must be coordinated with appropriate knowledge elicitation procedures. Without a model of expertise, a knowledge

engineer would be overwhelmed by the mass of unorganized entities which must be dealt with during knowledge acquisition. Without appropriate knowledge elicitation tools, establishing a complete knowledge base would be too time consuming. For the desired coordination, which should be performed during the domain definition (Woodward, 1990), the knowledge engineer must deal with the following three problems:

1) A good model of expertise must be found or developed. For the purpose of the present paper it is not critical which specific model-approach is selected (Karbach, Linster & Voss, 1990a), but only that the knowledge acquisition is model-based, i.e. that an adequate model is used at all. Such a model may be obtained by selecting it from a library of models (Breuker & Wielinga, 1989) or by constructing it, for instance through an ontological analysis (Alexander, Freiling, Shulman, Rehfuss & Messick, 1987).
2) Traces of expertise, which serve as information sources, must be found and adequate knowledge elicitation procedures are to be developed even for elicitimg implicit knowledge (Lewandowski, Dunn, & Kirsner, 1989) so that a complete knowledge base can be efficiently established.
3) Finally, the knowledge elicitation must be coordinated with the specifically selected model of expertise, so that the elicited knowledge appropriately supplements the model. In complex real world domains, a number of different traces of expertise will exist. Some traces of expertise may be generally accessible but incomplete, others may be confidential within some community and possibly contradictory between different competing communities. Therefore, it is quite critical to develop or select a model, so that the knowledge details required for some model can indeed be supplied.

In the current research, an integrated knowledge acquisition method is developed and applied. The proposed method, including the formation of a model of expertise as well as explanation-oriented knowledge elicitation procedures will be described after an analysis of historical, sociological and cognitive aspects of the application domain, which was motivated by general design requirements for knowledge-based systems (Gaines, 1989). Finally the results of a pilot application of the method will be reported and discussed.

2. ANALYSIS OF MECHANICAL ENGINEERING KNOWLEDGE

Our knowledge acquisition approach is part of the ARC-TEC-project (ARC-TEC stands for Acquisition, Representation and Compilation of TEChnical Knowledge; Richter, Boley & Wetter 1989). The application domain of the project is mechanical engineering and computer-integrated manufacturing where knowledge-based product models (Legleitner, 1990) are to be shared among different task applications (Bullinger & Salzer; 1989). However, for the purposes of the current paper, only Computer Aided Planning (CAP) for the manufacturing of workpieces (Thien-Chien & Wysk, 1985) and more specifically the manufacturing of rotational parts will be considered.

Rotational parts or workpieces are manufactured by putting some more or less cylindric piece of metal (mold) into the fixture (i.e. chucking) of a manufacturing machine (CNC machine). The chucking fixture together with the attached mold is then rotated at a relatively high speed, with the longitudinal axis of the cylinder as the rotation center. Simultaneously, a cutting tool moves along some contour. The rotational axis and all movements of a specific cutting tool (movements which perform a cut as well as movements which only position the tool) lie in a plane. As a result of this processing, axle shafts, drive shafts, or bevel wheels may be produced.

The technique of manufacturing a rotational part may be better understood by a comparison to pottery. The manufacturing processes are similar to making a pot in the following way: One puts or attaches a piece of clay to a potter's wheel and shapes the clay to a specific form, only by

removing some parts of the clay while the potter's wheel is turned[1]. Contrary to the soft clay, which also allows a potter to push some material to a neighboring position, a rotational part or workpiece (metals) is shaped solely, by removing materials with a hard cutting tool.

Figure 1 shows a graphical representation of a (partial) workplan for a rotational part. The geometric form of the mold and the target workpiece are overlayed. The specific chucking fixture is shown. The sequence of cuts is specified. For a complete workplan additional specifications are needed such as the cutting tools used.

While the principles of manufacturing a rotational part may be simple, the actual manufacturing process and consequently also the respective planning task is extremely complex. The complexity comes from the large number of different possibilities for each operation and the various dependencies among the different operations. An additional complexity in the planning process, arises through the large body of possible technologies, accumulated scientific knowledge and various practical experiences, which are potentially useful for improving or even optimizing the manufacturing process. The application of expert systems to this domain is generally considered promising (Richter, 1990; Pfeiffer, Siepmann & Teichmann, 1988).

2.1. Historical, Sociological and Cognitive Aspects (HISOCO-Analysis)

The following results show that a historical assessment of a subject domain can reveal the traces of expertise that have been established in an area. Thereby it may also be evaluated which changes may be expected in the near future. A related analysis of sociological factors determines where traces or expertise can be found, which traces are publicly available and which traces are treated confidentially for example within a company (Schmidt & Schmalhofer 1990).Cognitive analyses of human expertise indicate how planning problems are solved in practice and which principles should be satisfied for developing a cognitively adequate expert system and finding appropriate elicitation methods. Such a combined analysis of the historical, sociological and cognitive aspects of expertise in a subject domain is termed HISOCO-Analysis (Fischer & Schmalhofer, 1990).

Historically, mechanical engineering originated from craftsmanship. Over a time period of several centuries, craftsmanship (the competence of the community of craftsmen) developed into a differentiated set of specialized skills, which are now distributed among a number of differently educated people. In addition to the practical skills of manufacturing and efficiently preparing the manufacturing process (design of workpiece, workplanning etc), polytechnical research has provided a more general and theoretically oriented understanding. Today, practical skills as well as theoretical knowledge are available as two different traces of expertise in mechanical engineering (Spur, 1979; Fandel, Dyckhoff & Reese, 1990).

Whereas the results of theoretical research are frequently published and therefore generally accessible, the more practical skills are individually communicated in apprenticeships or confidential (Brödner 1985; Spur, 1979). Theoreticians, who are not directly interested in practicing engineering themselves, further their reputation when their results are published. Highly skilled practitioners, on the other hand, who earn a living with their practical skills, would create additional competition for themselves by widely distributing the specific tricks of their trade. Practical skills are therefore often idiosyncratic to a person or a company.

Since theoretical results are published and discussed in a larger community, they are more generally accepted and in themselves more consistently denoted. The transfer of theoretical knowledge into practical fields and the combination of theoretical knowledge with practical skills is a major challenge of the future which cannot be ignored in expert system research.

[1]This analogy was pointed out by Harold Boley

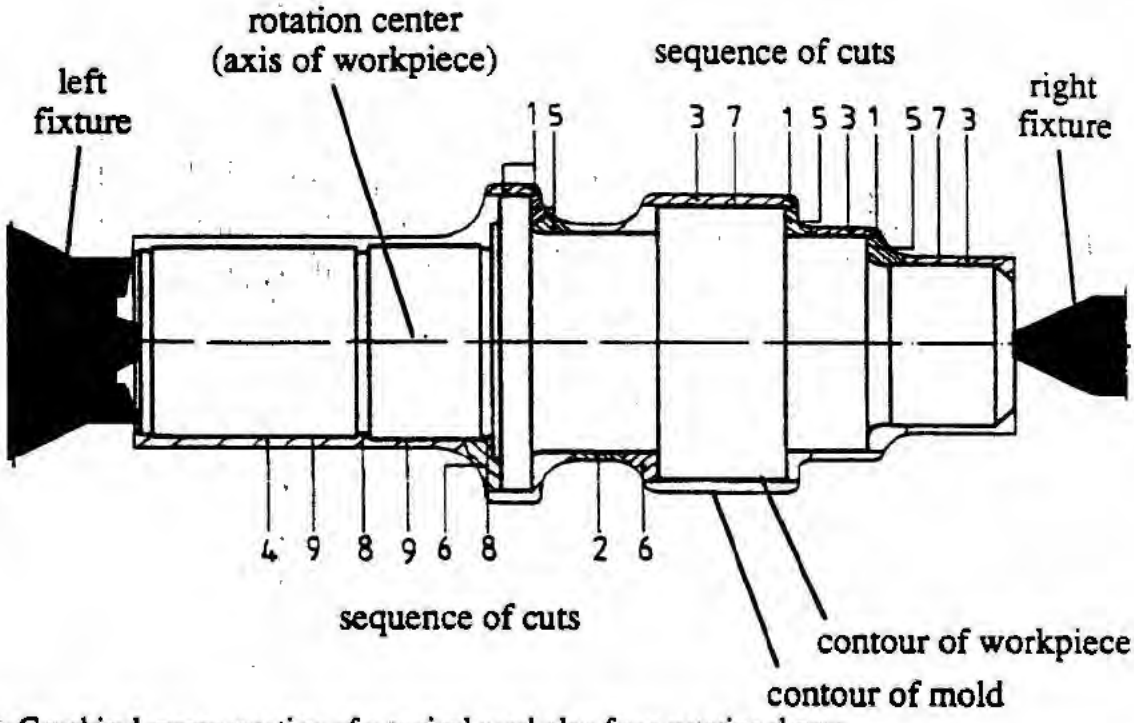


Figure 1: Graphical representation of a typical workplan for a rotational part

A frequently used planning procedure in mechanical engineering consists of adapting previous solutions to newly arising situations (Thoben & Schmalhofer, 1990). Cognitive analyses show that people form a library of experiences from cases (Shaw & Woodward, 1989; Schmalhofer & Wetter, 1988; Strube & Janetzko, 1990; Weber, 1988; Riesbeck & Schank, 1989). The practitioners memories consequently may contain a large library of experiences which are possibly hierarchically structured. Theoreticians on the other hand are usually concerned with more general rules.

The activities of practitioners and theoreticians yield different traces of expertise, which can be utilized for knowledge acquisition:

1) The general knowledge which renowned theoreticians accumulated in their research can be found in various books.
2) The specific solutions which practitioners have found over a number of years are stored in the filing cabinets or databases of companies. Sometimes selected cases have also been published.
3) An implicit classification of the various types of workpieces is supposedly stored in the practitioner's memory, which can be tapped with appropriate knowledge elicitation techniques.

Figure 2 shows an illustration of the three identified traces of expertise. These traces of expertise are partially overlapping and complement one another. Consequently, a more complete and qualitatively better knowledge base may be constructed with an integrated knowledge acquisition from the different traces of expertise.

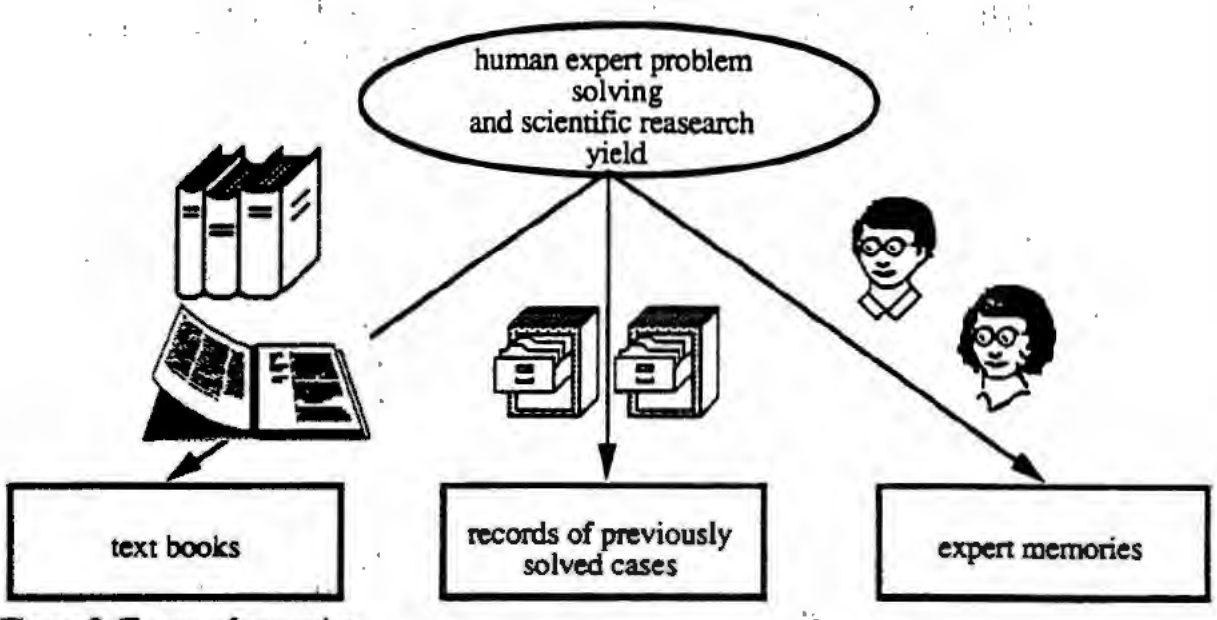


Figure 2: Traces of expertise

2.2. Model of Expertise for Production Planning

It has already been pointed out that a cognitive analysis of the human planning processes may also yield a high level model which can be used as a model of expertise for developing a knowledge based system. By building the model of expertise on a cognitive foundation, a user centered expert system may be developed which meets a higher acceptability in the professional community of the subject domain (Schmalhofer, 1987). A cognitively adequate model of expertise also facilitates the validation of elicited knowledge. When the elicited knowledge is classified according to the way of thinking in the specific subject domain, the expert will feel much more comfortable in validating the acquired knowledge as authentic expert knowledge.

In empirical investigations (Thober & Schmalhofer, 1990) and an additional think-aloud study (Schmidt, Legleitner & Schmalhofer, 1990) the expert´s problem solving behavior for finding a production plan was investigated in some detail. The empirical results showed that all production planning can be seen as modification planning where a more or less abstract skeletal plan is retrieved and subsequently refined to a specific workplan. When the skeletal plan is quite general, generative planning processes play the major role. When the skeletal plan is so specific that it is very close to a detailed production plan, the memory retrieval accomplishes the planning goal (replication planning). Replication planning is thus a special case of modification planning, where a complete plan is already available. Between these two extremes the various forms of modification planning occur. In addition, the planning processes take into consideration the specific machines which are available.

The abstract types of processing of the resulting model of expertise are shown in Figure 3; The problem of production planning consists in finding an adequate production plan for a given workpiece which is to be manufactured in some given factory. The description of the workpiece is presented by the workpiece model and the description of the factory (i.e. the available machines and tools) in the factory model. From these concrete data an abstract feature description of the workpiece and an abstract context specification are obtained through the application of abstraction or classification rules. To these abstract workpiece and context descriptions a skeletal plan which contains an abstract description of a concrete production plan can be associated. The skeletal plan is refined with the help of the workpiece and the factory models so that an executable production plan is obtained. Supposedly an expert´s memories include a hierarchy of workpiece classes. Results from cognition (Chi, Feltovich & Glaser, 1981; Egan & Schwartz, 1979; Glaser & Chi,

1988) provide substantial evidence that these classes have been formed according to skeletal production plans.

In our knowledge acquisition method, this model is used for immediately classifying each knowledge item into the different categories provided by the model. The categories are determined through the different abstract types of processing (i.e. abstracting, associating, selecting, refining information) expressed in the model of expertise. Because of the high complexity of the domain, a domain-specific decomposition of the task into a small number of canonical subtasks was furthermore introduced. In particular, the cognitive analysis showed that production planning for rotational parts can be subdivided into the subtasks of 1) finding a chucking fixture, 2) determining the sequence of cuts, and 3) selecting appropriate cutting tools. Through this additional subdivision of each category of the model of expertise, one obtains knowledge classes with a reasonable number of knowledge units. These sizeable categories facilitate knowledge acquisition. This is particularly important when knowledge is to be elicited from several traces of expertise.

3. DESCRIPTION OF INTEGRATED KNOWLEDGE ACQUISITION

3.1. General Problems

The problem of combining information or knowledge arises in two different situations:

When new or additional information (or knowledge) is to be inserted to an already existing knowledge base or some model of expertise, the new information needs to be adjusted to the givens of the existing structure. This process may be generally called knowledge assimilation (Lefkowitz & Lesser, 1987).

Conversely, when knowledge is independently acquired from different but equally respectable sources (e.g. different experts, different traces of expertise, etc.), from different representations, or with different and equally profitable tools, knowledge may be integratively analyzed. Thereby, a single uniform knowledge base can be produced. Such a merging of knowledge may be termed integrated knowledge acquisition (Brazdil & Torgo; 1990; Morik, 1989). Although aspects of knowledge assimilation can also be found in this paper (e.g. in the insertion of the elicited knowledge into the model of expertise), its major focus is the integration of knowledge from different sources.

After several profitable traces of expertise have been identified, an integrated knowledge acquisition can accomplish an efficient utilization of the different traces and an early validation of the elicited knowledge. Usually, these traces will contain overlapping information. By itself none of the traces will provide all the relevant information. However, the combination of several traces together with the model of expertise may be sufficient. In its combination the provided information may appear or actually be contradictory. Consequently, an integrated knowledge acquisition method should address the following problems:

1. Are there parts of an information source which can be described as redundant or contradictory with respect to other knowledge and how can such redundancies and contradictions be eliminated?

2. Which information is relevant for solving the target problems and which information is in addition needed for the combined information to be sufficient?

The general problem of integrated knowledge acquisition thus consists of finding a set of knowledge units which is sufficient for performing the target tasks. In addition, the set of units should not be arbitrarily redundant, i. e. it should satisfy some weak minimality requirements.

From a formal point of view, it is well understood that in general the specified problems are intractable if not undecidable. From a practical point of view, it is equally obvious that the specified problems must be treated in a sophisticated manner, so that at least humans can eliminate major contradictions and undesirable redundancies.

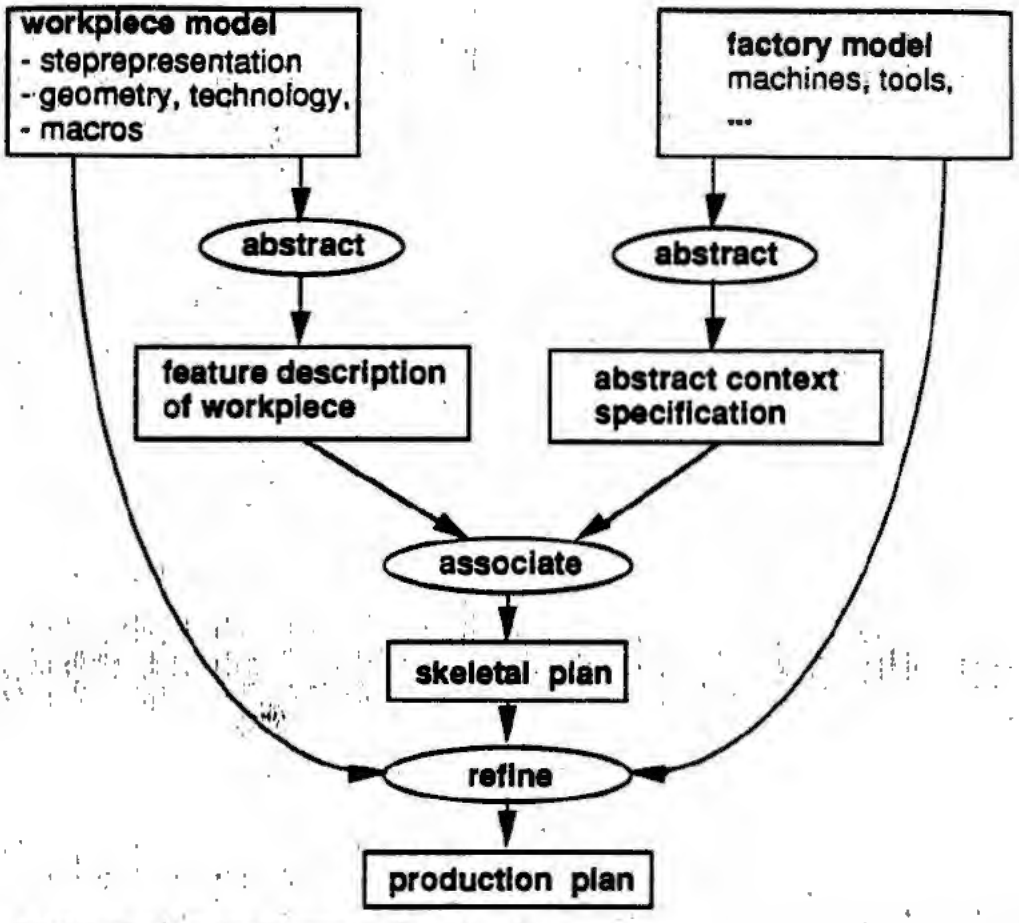


Figure 3: Model of expertise for production planning

3.2. Basic Principles

At first, the principles for assessing the relevance and sufficiency of knowledge will be introduced. Secondly, the principles for eliminating contractions and undesirable redundancies.will be discussed. Consequently, it will be shown how the application of these principles yields an early knowledge validation.

3.2.1. Relevance and sufficiency assessment: The relevance and the sufficiency of general knowledge units can be determined by having an expert explain prototypical target tasks (cases) with the general knowledge units. These units (e.g. general statements from a text or general experiences from human experts) are elicited, because of their great generality and high reliability. High quality task solutions are selected, so that the specific set of examples somehow forms a base for all problems which the expert system is supposed to solve. Because the set of examples defines the particular area of competence of the expert system, they can be used as a guide for selecting those general knowledge units which constitute the relevant knowledge. By having the expert instantiate the general knowledge with respect to the specific cases and by performing constructive explanations, a bridge is built between the general knowledge units and the specific cases. Thereby a knowledge integration can be performed between a case base and a text, or a case base and general expert experiences. Relevance and sufficiency are consequently assessed with respect to the selected cases.

3.2.2. Elimination of redundancies and contradictions: In order to eliminate major redundancies and overt contradictions the various knowledge units must be compared in a systematic way. The model of expertise together with the domain oriented decomposition into

subtasks provides a useful categorization for the elicited knowledge units. By comparing the knowledge units of a category with all other knowledge units of the category the most critical redundancies and contradictions are eliminated.

3.2.3. Early knowledge validation: Through the application of these principles an early knowledge validation is performed. With informally presented knowledge (e.g. natural language sentences and rules), which is often more comprehensible to a human expert (Schmidt & Wetter; 1989) relevance, sufficiency, redundancies and contradictions can be identified relatively soon. Formal considerations (i.e. completeness and consistency of knowledge base) are thereby already treated with informally denoted knowledge. Performing such formal considerations at an the informal level is termed early knowledge validation.

3.3. Acquisition Method

The proposed integrated knowledge acquisition method basically consists of four episodes, which can be flexibly interspersed. Whereas the first three episodes (1. explanation of solved cases, 2. comparison of similar or related knowledge elements, 3. competence delineation) concern the elicitation and analysis of knowledge at an informal level, the forth episodes is concerned with the formalization of that knowledge (4. formalization phase). The first three phases will be described in some detail while he formalization phase will be only outlined. An overview of the acquisition method is presented in Figure 4.

3.3.1. Explanation Episode: In this phase, information which is relatively general and supposedly relevant for the target tasks of the future knowledge based system is selected from one or several appropriate sources. Independent from this selection of general information, a set of prototypical previously solved cases is determined. Other than through the desired competence of the future expert system, the general information and the set of cases are independently selected from one another, so that the two information sources can independently confirm each other. This is of crucial importance for the early validation.

The major task for the domain expert in the explanation episode consists in applying the selected general information to the previously solved cases by explaining the solution cases according to the structure of the model of expertise. When the model of expertise exceeds some degree of complexity, it is desirable (or inevitable) to decompose the explanation task into a number of explanation subtasks (Simon, 1981). For each explanation subtasks a different information source may be used and the combination of the different explanation subtasks should be overlapping. For instance, with the described model of expertise (see Figure 3) an expert's general memories about the specific problem classes in the domain may be used for the subtask of defining abstraction rules by explaining the various problem classes (Bergmann & Schmalhofer; 1990). Additionally, theoretical knowledge from text books may be mainly used for the subtasks of finding the associate and refine rules (Schmidt & Schmalhofer; 1990). Through the model-oriented explanation of each case, the completeness of the general information with respect to the specific case is established, and it is assimilated to the model of expertise.

In order to further facilitate the examination of the sufficiency of the elicited knowledge and its assimilation to the model of expertise, each elicited knowledge unit is put into a specific form (e.g. an IF- THEN form may be used, when a formal rule base is to be established later on). Since the model of expertise has been constructed to be cognitively adequate, the human expert should feel comfortable in using this structure for generating a respective explanation.

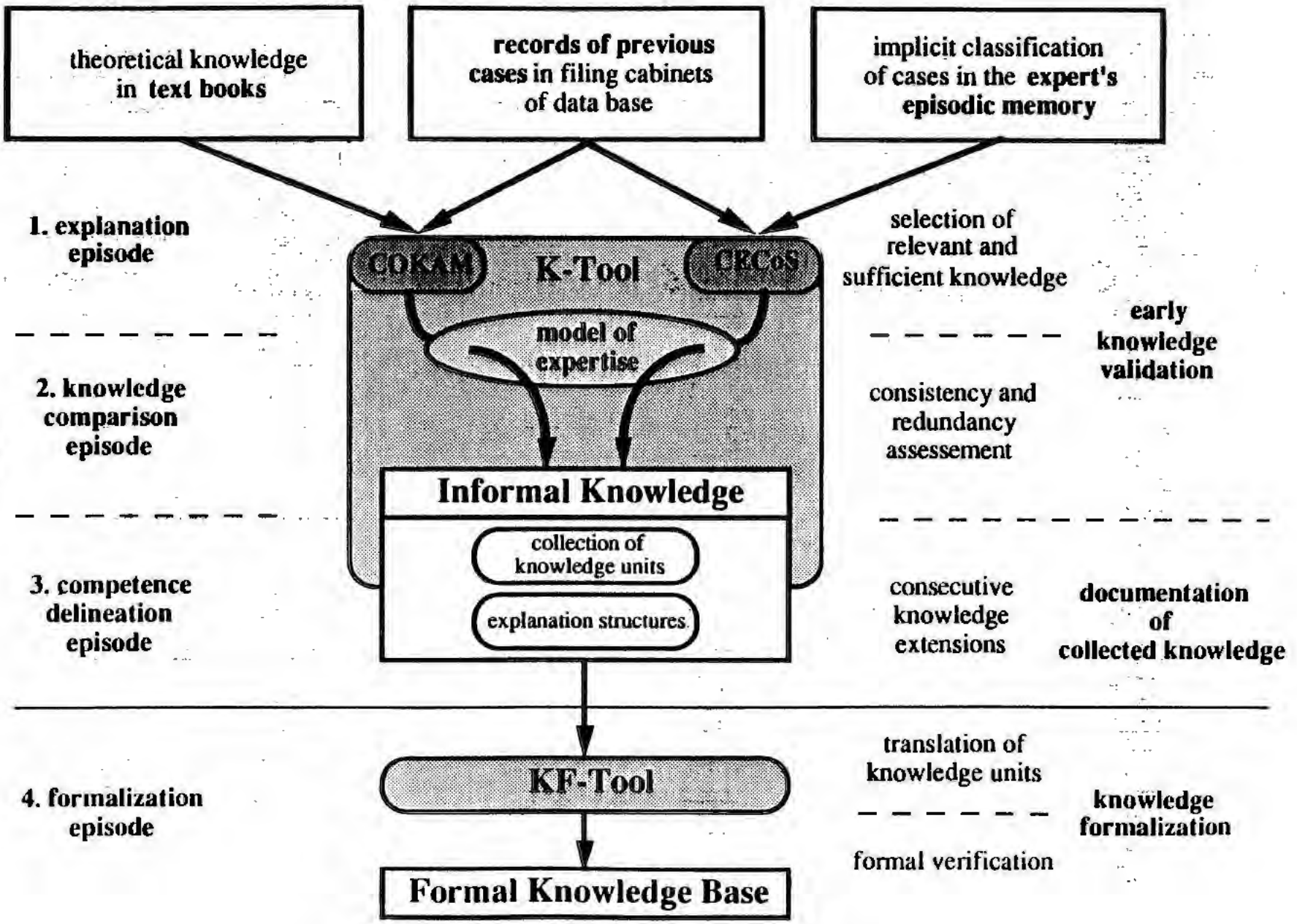

theoretical knowledge in text books
records of previous cases in filing cabinets of data base
implicit classification of cases in the expert's episodic memory
1. explanation episode
COKAM
K-Tool
CECoS
model of expertise
selection of relevant and sufficient knowledge
early knowledge validation
2. knowledge comparison episode
consistency and redundancy assessement
Informal Knowledge
collection of knowledge units
explanation structures
3. competence delineation episode
consecutive knowledge extensions
documentation of collected knowledge
KF-Tool
translation of knowledge units
knowledge formalization
4. formalization episode
formal verification
Formal Knowledge Base

Figure 4: Integrated knowledge acquisition method

3.3.2. Knowledge Comparison Episode: After two or more of the solution cases have been explained, the knowledge units in the different categories of the model can be compared to one another. During this comparison, some knowledge units may be found to be redundant and consequently eliminated, others may be found to be generalizable, still others may require a differentiation. If a contradiction is found, one or several explanations may have to be revised. This may result in an elimination or adjustment of some knowledge units The number of times a knowledge unit is used in different explanations may additionally be counted for each knowledge unit. Each knowledge unit is stated as general as possible and as specific as necessary for explaining the different cases.

3.3.3. Competence Delineation Episode: In this episode the possible competence determined by the already established (informal) knowledge base is to be delineated. In a most conservative assessment, it is noted that the acquired knowledge is sufficient for successfully solving those problems, whose solutions were used for knowledge elicitation. In particular, if these solutions were removed from the knowledge base, the respective problems may still be solved by appropriately combining the stored knowledge units. One such combination is the explanation structure which the expert generated and which is stored as part of the informal knowledge base. The knowledge base can thus be used to solve problems by re-generating previous solutions (e.g. for replication planning).

The acquired knowledge may in addition be used to solve problems which at some level of generality are structurally identical to the prototypical problems used in the knowledge acquisition phase. This can be accomplished by following a stored explanation structure. Thereby the general knowledge units are appropriately instantiated with respect to the actual problem, which may be more or less similar to the previously considered prototypical cases. The knowledge base can now be used to solve problems by adapting previous solutions (e. g. for modification planning).

By decomposing the expert´s explanation structures into meaningful segments, solution methods for various subtasks may be identified. By combining solution methods from the different previously treated problems, solutions to structurally new problems may be created (e.g. for novel planning tasks).

The largest possible area of competence which is based on the acquired knowledge may be delineated by applying the knowledge units in the most general way and by combining the knowledge units in all possible ways. For obvious reasons, for the largest possible area of competence only a rough and pragmatically oriented assessment may be even attempted.

The described competence delineation allows to determine whether iterations of the previously performed knowledge acquisition episodes should be performed or whether the knowledge base already satisfies the customer´s requirements (i.e. the requirements of the intended user of the expert system).

3.3.4. Formalization Episode: In the formalization episode, each knowledge unit is one by one translated into its respective formal representation and stored in the formal knowledge base. Thereafter the previously informal explanations are now formally examined by traversing the stored explanation structures, this time however with the formalized knowledge units. For those explanation structures, which can be successfully traversed, the early knowledge validation is formally verified. Unsuccessful traversals indicate a bug in the execution of the knowledge acquisition, e. g. insufficiencies in the informal knowledge base, errors in the translation from the informal to the formal knowledge units, etc.

3.4. Case Oriented Knowledge Elicitation with COKAM and CECoS

For the integrated knowledge acquisition, knowledge elicitation tools or procedures are required which select the relevant information from the various traces of expertise. COKAM and CECoS are such knowledge elicitation tools or procedures, where each of the procedures performs a joint elicitation of two traces of expertise. COKAM performs a knowledge acquisition from texts,

which is enriched by utilizing records of solved cases. With CECoS previously recorded problem solutions are combined with an expert's high level understanding of the global structure of a task domain. Both procedures can be applied to perform the explanation episode of the integrated knowledge acquisition. Basically each procedure consists of two phases, a selection and an application phase. In the selection phase, it is first determined which sample of a trace of expertise is to be elicited before the respective elicitation is conducted. In the application phase an explanation task is to be performed which jointly utilizes both traces of expertise. The application phase yields the informal knowledge. Figure 4 indicates the role of COKAM and CECoS within the framework of the proposed integrated knowledge acquisition. A brief characterization of COKAM and CECoS will be presented next.

COKAM: In the selection phase of COKAM an expert has to select texts and text segments which in his or her opinion contain relevant information for performing the target tasks of the future expert system. Independent of the expert's selection of text, the knowledge engineer selects previously solved cases from a filing cabinet or a data base. In the application phase the expert has to explain each solution case with the help of the selected text segments and his common sense knowledge. The common sense knowledge is thereby used to fill the information gaps of the collection of text segments. In a previous application, COKAM was used to elicit knowledge without a model of expertise (Schmidt & Schmalhofer, 1990). However, the integrated knowledge acquisition method requires that the elicited knowledge is categorized according to the model of expertise, which may be accomplished with the extended version COKAM$^+$.

CECoS: When CECoS is applied together with COKAM, the same cases should be used. In the second step of the selection phase, CECoS performs a hierarchical classification of problem classes by eliciting global judgements from human expert memories. For example, after a complete paired comparison of the cases by the expert, a hierarchical order of problem classes may be obtained by a hierarchical cluster analysis. In the application phase, the expert has to generate appropriate feature descriptions for each class, so that he can explain the various class memberships of the cases and the different class subsumptions.

4. FIRST RESULTS OF A PILOT APPLICATION

Naturally, knowledge acquisition in a complex real world domain is more demanding and more time consuming than in limited test domains. Nevertheless, it is important to assess the usefulness of the developed method relatively soon. Therefore, a pilot application was performed, so that some aspects of the proposed knowledge acquisition method could be evaluated. In particular, it was investigated how willingly experts would relate general statements of expertise (text, memories) to specific cases by generating an explanation with them. Secondly, it was to be determined, whether sufficiency and relevance can be judged by early knowledge validation. More specifically, it was tested whether appropriate explanation structures could be elicited with COKAM and CECoS and whether the identified knowledge units could be assimilated to the model of expertise.

The two knowledge elicitation procedures COKAM and CECoS are currently being implemented. Preliminary implementations and paper and pencil surrogates were used to perform the pilot application. For these tests, five work pieces with their corresponding workplans were selected from a catalogue (SPK, Feldmühle). The technical drawings of the selected shafts served as a graphical representation of the respective product models. A graphical representation was also used for the workplans which showed the workpiece overlayed with the mold, the used chucking fixture and the sequence of cuts.

4.1. COKAM

Since the task of explaining the whole workplan is rather complex, the pilot application of COKAM was limited to the elicitation of the knowledge which is relevant for the selection of a chucking tool. The text which was used as one trace of expertise described the different chucking

tools which are available and the criteria which must be considered for the selection of an appropriate chucking tool. The expert had no problems identifying text segments which he considered to contain relevant knowledge units. The identified text segments were decontextualized, i.e. references to previous text were resolved so that independent knowledge units were obtained for the informal knowledge base (anaphora resolution). The following items are examples of such knowledge units[2] :

1. For long workpieces two chucking fixations are necessary.

2. When an axial chucking is used for long workpieces, the workpiece may buckle.

4.1.1. Explanation of solved cases: In COKAM, the expert was to explain a particular case by using the identified knowledge units as far as possible. It was found that the knowledge units from the text were not sufficient to explain the solutions to the particular cases, e.g. why lathe center chucking was used for the manufacturing of a specific axle shaft. A complete explanation could be achieved, however, by the addition of some common sense knowledge such as:

3. Center holes are possible, when smooth vertical planes on the two ends of the workpiece are not required.

These units of common sense knowledge were also added to the informal knowledge base. It was found that for subsequent cases less and less common sense knowledge units had to be added. A limited number of knowledge units thus appears to be sufficient for the explanation of a large number of cases. Some of the knowledge units which were extracted from the text were not used in any of the five explanations. The expert deliberated that these units were subsumed by some other knowledge units or were of no use for the specific type of problems, i.e. selection of chucking tools. Using knowledge units from a text for the explanation of individual cases is apparently suited for testing the relevance and sufficiency at an informal level.

4.1.2. Assimilation of the knowledge units into the model of expertise: The elements of the informal knowledge base were then assigned to the categories in the model of expertise. The knowledge unit 1. is a typical association rule, since it relates a feature of the workpiece (long) to a feature of the workplan (two chucking fixations). The knowledge unit 3. constitutes a common sense explanation given by the expert. Since it relates one feature of the workpiece (smooth vertical planes on the two ends of the workpiece are not required) to another one (center holes are possible) from which consequences for the workplan can be derived more easily, it can be classified as an abstraction rule.

A categorization of the remaining knowledge unit is not so obvious. For the knowledge unit 2. problems arise, since it contains two premises one of which refers to a feature of the workpiece whereas the other refers to a feature of the workplan. Furthermore, the conclusion (the workpiece may buckle) is only a warning which must be taken into account in the planning process. Further analysis revealed that this knowledge unit constitutes a refinement rule. It can be reformulated as:

2a. If an axial chucking is suggested by the skeletal plan and the workpiece is long, then the other parameters of the workplan (e.g. chucking power) must be determined in such a way that the workpiece does not buckle.

This reformulation makes obvious, how unit 2. is understood by the expert. Statement 2a. makes explicit some of the knowledge which had been compiled out in statement 2. (Anderson, 1983). In other words, the knowledge of statement 2a is implicitly contained in statement 2.

In order to avoid such problems with the categorization of the knowledge units, the knowledge elicitation procedure COKAM was modified. The modified version COKAM$^+$ differs from the original version in that the expert is told to use the model of expertise as a guideline when generating explanations for the individual cases.

---

[2]The following items are translations of the original German sentences.

4.2. CECoS

For a first test, the same five cases were used as in the pilot application of COKAM. All possible pairs of product models were randomly presented on the screen. In the first elicitation task the mechanical engineer (expert) had to assess for each pair the similarity of the respective production plans. The expert indicated the similarity by a rating between 1 and 7. With the $n*(n-1)/2 = 10$ collected similarity ratings CECoS performed a hierarchical cluster analysis, resulting in a tree, which is shown in Figure 5. Through this tree an initial hierarchy of production classes for product models is established, where each nonterminal node indicates a class. The terminal nodes stand for the prototypical members of the class.

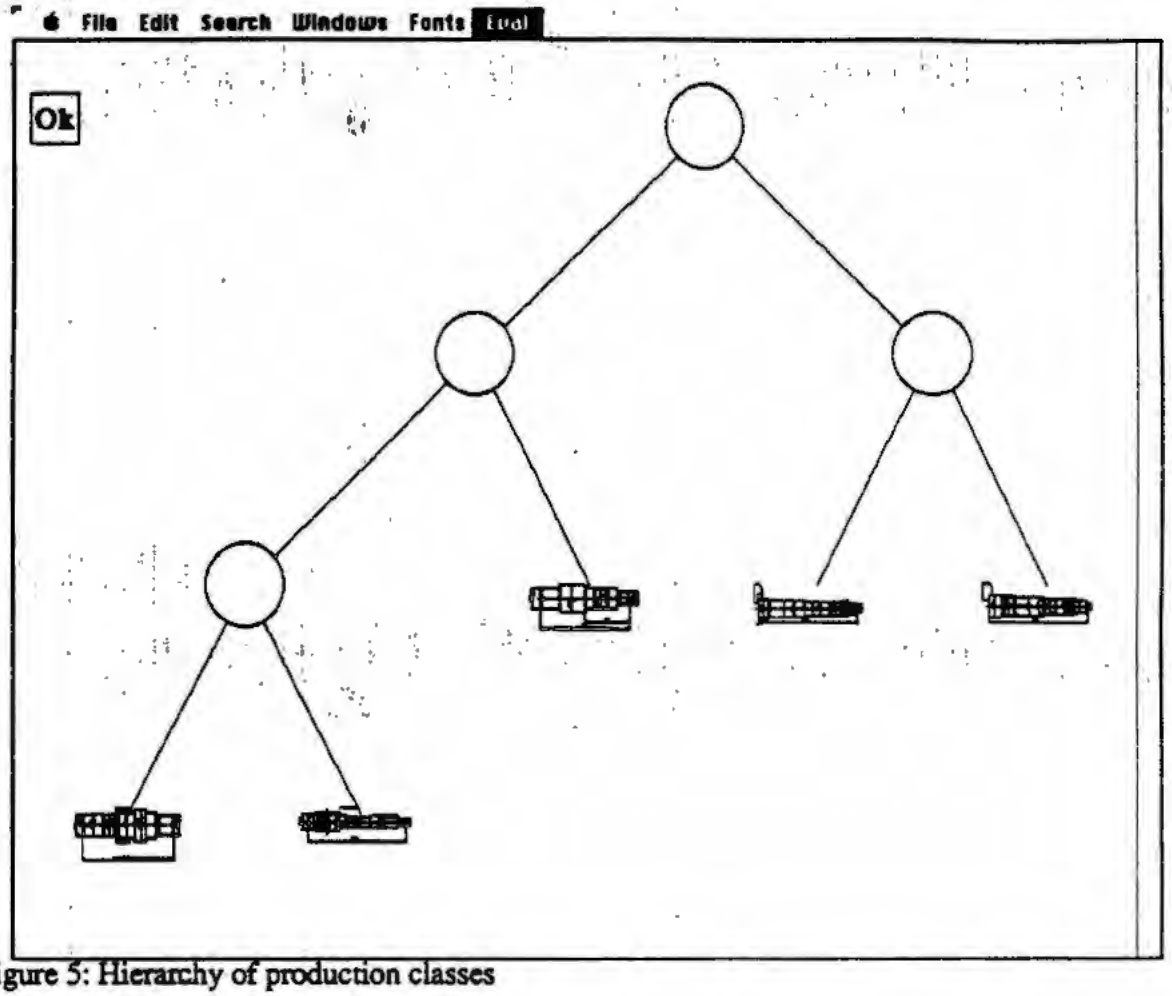


Figure 5: Hierarchy of production classes

4.2.1. Explanation of production class hierarchy: The generated class hierarchy was explained in the application phase of CECoS. For each class of the hierachy the expert was to name characteristic features which all workpieces in the respective class have in common. The named features should then be defined in terms of data given in the workpiece (or factory) model.

For example, the following features were generated by the expert:

1. *workpiece of medium size*
2. *monotonic contour of workpiece*
3. *workpiece behaves stable during production*
4. *two chucking fixations necessary*
5. *can be manufactured with low cutting speed only*

Some features like 1. and 2. were defined rather readily, because they can be easily related to geometry or technology primitives of the workpiece model. A workpiece of medium size, for example, was defined as a workpiece with a length of 70 to 800 millimeters and a maximal diameter of 10 to 300 millimeters.

Features which referred to the mechanical properties of the workpiece during the production process were more difficult to define. For example, the feature 3. had to be defined by a relatively complicated combination of terms such as length of workpiece, minimal diameter of workpiece and material of workpiece.

Features 4. and 5. directly refer to a possible manufacturing plan of the workpiece. The expert showed some difficulties when defining them in terms of the workpiece model.

4.2.2. Assimilation of the defined features into the model of expertise: The features 1. to 3. are typical abstraction rules which relate workpiece data to more abstract workpiece features. Knowledge of this category is typically obtained by applying CECoS. According to the model of expertise, the features 4. and 5., which are workplan features, should not be directly related to workpiece data. Instead they should be associated with workpiece features which are abstracted from workpiece data. This assumption fits well to the observation that the expert could only define these features in terms of the workpiece model after performing COKAM. For example, the feature 4. is associated with a workpiece feature by knowledge unit 1. elicited by COKAM:

For long workpieces two chucking fixations are necessary.

This workpiece feature (long) must be defined in terms of the workpiece model. Such a definition could be obtained through CECoS by having the expert assign this feature to the most general production class in the hierachy. It is therefore suggested to apply COKAM before CECoS. Thereby the relevant features can be identified by COKAM and later defined through CECoS. CECoS is particularly suited for elicitating these abstraction rules.

The pilot applications of the knowledge acquisition method also showed that COKAM reminds the expert of the relevant terms: when CECoS was performed after COKAM, the expert was more productive in generationg good feature descriptions in CECoS.

5. DISCUSSION

When inference engines were extracted from the first successful expert systems (e.g. MYCIN), for application in another domain, knowledge editing and knowledge elicitation tools proved to be useful. In those days, knowledge acquisition was tied to the structure of the inference engine, whereas the cognitive structures of human problem solving in the particular domain were ignored.

It was then believed that the knowledge acquisition problem would consist in uncovering unwieldy knowledge (e.g. implicit or compiled knowledge) which requires sophisticated elicitation and analysis methods (repertory grids, think aloud methods, interviewing techniques together with scaling techniques, protocol analysis, hierarchical cluster analysis). Different types of such elicitation and data analysis procedures were consequently combined to knowledge elicitation workbenches (e.g. Diederich, Linster & Ruhmann & Uthmann., 1987) so that a more complete knowledge collection would be established.

The success of these elicitation procedures has mainly been restricted to relatively small tasks where an expert system can be developed within a few hours (Boose, Bradshaw & Kitto, 1989). Without a sophisticated model of expertise, however, such elicitation techniques can hardly be successfully applied to a complex task domain, such as production planning in mechanical engineering.

The limitations of bottom-up elicitation procedures can be overcome with the help of sophisticated models of expertise. The assumption that the knowledge acquisition problem consists in finding

an appropriate model was very fruitful. Through an epistemoligical analysis of existing expert systems Clancey (1985) identified heuristic classification as the prevalent problem solving method in successful expert systems. Chrandrasekaran (1986) defined various generic tasks. In the KADS projects (Breuker & Wielinga, 1989) a library of interpretation models is being developed. Through such models, expert systems can be specified at the knowledge level (Newell, 1982; Clancey, 1989).

Such knowledge level descriptions of expert systems can be used to elicite exactly that knowledge which can be assimilated to the model so that the model together with the assimilated knowledge are sufficient for performing the target task. In order to develop a successful expert system, it must furthermore be warranted that appropriate traces of expertise and corresponding elicitation tools are available and accessible to the knowledge engineer.

Frequently, alternative models (opportunistic planning, case-based planning, skeletal plan refinement, etc.) are available for the same task. In these situations, that model may be selected which is cognitively most appropriate and for which supplementary traces of expertise are readily available. Knowledge elicitation and model building can thus be coordinated. This is accomplished by the HISOCO analysis: the cognitive analysis assists in specifying the best model of expertise and the sociological analysis reveals the most appropriate traces of expertise in the domain.

In the reported research we have analyzed the complex task of production planning and established a knowledge level description for that task. The knowledge level description was obtained within the KADS framework. In KADS terminology, a task structure and an inference structure were established for production planning (Kühn, Schmalhofer & Schmidt, 1990). The task structure describes that each workpiece is first classified into some category that is associated with a skeletal plan. Supposedly the skeletal plan is appropriate for building a complete plan through refinement operations. In case of an inappropriate selection of a skeletal plan these procedures can be iterated. These task structure lead to the specification of the inference structure which was used as the model of expertise (see Figure 3). Because the inference structure provides the most detailed categorization of the domain knowledge, it was used for the knowledge acquisition rather than a complete interpretation model.

Since production planning is a very complex process, three additional domain categories (chucking, sequence of cutting operations and tool selection) were used in addition to the inference structure as orthogonal subdivisions to further modularize the knowledge elicitation. After an analysis of the available traces of expertise had been performed, two appropriate knowledge elicitation procedures were defined.

Through a coordinated knowledge elicitation from texts, previously solved cases, and expert memories, an early knowledge validation was accomplished by having the expert cognitively walk through the solved cases (Lewis, Polson, Wharton & Rieman, 1990) by explaining them with the general information elicited from the other traces of expertise. This procedure should assist in developing knowledge bases which are easy to explain (Swartout, 1983).

5.1. COKAM and its Relation to Language Understanding

COKAM requires the human user to perform a number of operations in a particular sequence. The operations and their sequence are similar to the cognitive processes which a human may perform when he studies a text for extracting specific task knowledge. In particular those text segments which appear to be relevant, are selected for further processing. After appropriate anaphora resolutions have been performed, the selected knowledge units can be assimilated to some prior knowledge structure until all slots of the knowledge structure are filled from the text, from hypotheses or through common sense. Text segments which fit into the prior knowledge structure are thus represented at a different level (Schmalhofer & Glavanov, 1986) and encounter a lot of processing while other text segments are relatively fast forgotten (Kintsch, Welsch, Schmalhofer & Zimny, 1990). COKAM thus utilizes results from research on human text comprehension together with respective models on human text comprehension (Kintsch, in press). A successful

assimilation of natural language text into a knowledge structure may require many attempts which may produce several versions of the same text segment. Such processing can be performed with a previously developed tool. (Schmidt & Wetter, 1989).

COKAM also shows a number of similarities to other knowledge acquisition methods from text (Anjewierden, Wiedemaker & Toussaint, 1990, Möller, 1988; Szpakowicz, 1988; Jansen-Winkeln, 1988). The system of Szpakowicz for example supports semi-automated acquisition of structures from technical texts. It presumes an initial general model according to the KADS-method and then acquires conceptual structures from texts. The operator supports the semi-automated system, if it cannot make decisions on its own. The SHELLEY of Anjewierden provides a hypertext-based protocol and concept editor, where the words of the stored text are mouse-sensitive, so that the definitions of the words can be inspected. Also, annotations are stored so that remarks from different people can be read on demand.

5.2. CECoS for Retrieving Expert Memories

Not every detail of a previously solved case is stored in human memory. Instead the human memory forgets the details and remembers the essential characteristics of a case. Contrary to human novices, who judge the similarity of two or more cases according to superficial features of the case, experts judge the similarity by the solution method which is applicable to the case (Chi, Feltovich & Glaser, 1981). CECoS uses this fact to at first obtain an extensonal definition of the various production classes with the particular cases used. Thereafter an intensional class definition is obtained by having the expert generate appropriate feature descriptions for each class.Since an intensional description can be applied for any case, it is more useful for determining the class membership of a new case. The knowledge elicitation of CECoS is thus also adjusted to the properties of human memory.

5.3. Joint Application of both tools and integration aspects

The joint application of COKAM and CECoS allows to integrate two traces of expertise which contain general information with a collection of specific cases. Similar to humans who can learn from general statements and from specific examples by relating the general information to the specifics of the examples, both traces of expertise are used for the purpose of knowledge acquisition. This is accomplished by having the human exert explain the specific examples with the general information. This procedure has been shown to be quite useful for knowledge acquisition (Martin & Redmond, 1989). The knowledge integration is thus similar to human knowledge integration, where the integration is performed through an assimilation to the same model of expertise.

The elicitation of knowledge from multiple traces of expertise is also similar to KRITON (Linster, 1988) which consists of three tools, one used for protocol analysis, one used for interview analysis and a tool for text analysis. However there are also important differences: Whereas COKAM and CECoS provide a clearly defined procedure for combining the different knowledge sources, KRITON is open to all possible combinations. On the other hand, COKAM and CECoS are specifically tailored to a domain, where well thought through cases with their solutions as well as theoretical texts which relate to the cases are available. Contrary to KE workbenches, which provide the KE with tools but little guidance in how to use these tools, the proposed method supplies some machinery which guides the KE and the expert to the desired expert system, while still providing him with enough freedom on which knowledge acquisition episode to perform next. Similar to the ontological analysis (Alexander, Freiling, Shulman, Rehfuss & Messick, 1987) the proposed procedure thus provides a useful guideline to the knowledge engineer how to start the knowledge acquisition. This guideline has been particularly tailored to complex real world domains.

Eggen Lundteigen & Mehus, (1990) have recently proposed a computational model for integration, where checking for completeness, correctness and consistency where described as the main objectives of the knowledge integration process. In the current method, these objectives are accomplished in two subsequent phases. In the explanation episode the relevance and

completeness of the elicited knowledge is establish by having the expert give sufficient explanations for the various previously solved cases. A consistency and redundancy assessment is supported, by having the expert compare all knowledge elements of the various categories. This is made feasible through the division of the knowledge units into sizeable categories.

5.4. Case-Based Planning versus Skeletal Plan Refinement

Since our model of expertise is skeletal plan refinement and not case-based planning, cases are only used as a vehicle in knowledge acquisition and are not themselves stored in the knowledge base. The proposed knowledge acquisition method may therefore be described as case-oriented: it uses cases to elicit knowledge from the expert memories and for testing the sufficiency of the general knowledge from the text and the common sense rules expressed by the expert. This corresponds to the way in which experts often explain their knowledge. Contrary to case-based planning (Riesbeck & Schank, 1989), where individual cases are appropriately indexed and then stored in a case-base, the knowledge base of the intended system would not contain any cases. Instead, the cases are only used during knowledge acquisition (Fox, J., Mypers, C. Greaves & Pegram, 1985; Kidd, 1985) to find the relevant general knowledge units, which are sufficient to solve the target problems

5.5. Early Validataion versus Early Formalization

The current research can be seen as an attempt to apply the KADS methodology to a highly complex application domain. Within this framework, we were successful in specifying an interpretation model, by adjusting the skeletal plan refinement model to the requirements of mechanical engineering. In particular the different available machining centers had to be taken into consideration. Recent research within the KADS-community is developing formal foundations and logical specifications for describing the conceptual model (Wetter, 1990). The approach of KADS II and related work, thus proposes to formalize models relatively soon while the validation of the acquired knowledge is of less concern. In the current research, on the other hand, it is proposed to formalize knowledge only after a validation and documentation of the acquired knowledge has been performed. Through early knowledge validation, relevance, sufficiency, redundancy and consistency may already be assessed with informally represented but model-centered knowledge.

In a complex domain, eliciting the relevant knowledge turns out to be a problem, even when a good model of expertise is found. With the described HISOCO analysis and the model-centered knowledge acquisition method, we hope to master these challenges and to develop an easy to explain and easy to maintain knowledge-based system in a real world domain. Similar to OPAL (Musen, Fagan, Combs, Shortliffe, 1987) the proposed method also is domain specific and uses skeletal plan refinement. It is different from OPAL in that it does not use as specific a domain model. Although the integrated knowledge acquisition method was developed for the domain of mechanical engineering, one may therefore expect that it could also be successfully applied in other domains.

Acknowledgements

We greatly appreciate the cooperation of our domain expert Dipl.-Ing. Ralf Legleitner and of our collegues in the ARC-TEC project. Special thanks are due to Ralph Bergmann for his assistance in elaborating our ideas.

References

Alexander, J.H., Freiling, M.J., Shulman, S.J., Rehfuss, S., and Messick, S.L. Ontological analysis: an ongoing experiment. *International Journal of Man-Machine Studies 26* (1987), 473-485.

Anderson, J.R. The Architecture of Cognition, Harvard University Press, *Cognitive Science Series* (1983).

Anjewierden, A., Wielemaker, J., and Toussaint, C. Shelley - Computer Aided Knowledge Engineering. In *Knowledge Acquisition via Knowledge Integration*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., IOS Press, May 1990, pp. 41-59.

Bergmann, R. and Schmalhofer, F., CECoS: A Case Experience Combination System for Knowledge Acquisition for Expert Systems, *Tech. Rept. Interner Bericht des ARC-TEC Projektes, German Research Center for Artificial Intelligence GmbH*, 1990.

Boose, J.H. A Survey of Knowledge Acquisition Techniques and Tools. *Knowledge Acquisition 1*, (March 1989), 3-38.

Boose, J.H. and Bradshaw, J.M. Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge based systems. *Int. Journal of Man-Machine Studies 26*, (1987), 3-28.

Boose, J.H., Bradshaw, J., Kitto, C.M., and , Shema,D.B. From ETS to AQUINAS: Six Years of Knowledge Acquisition Tool Development. In *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, October 1-6 1989, pp. 5-1-5-17.

Boose, J.H. and Gaines, B.R. Knowledge Acquisition of Knowledge-Based Systems: Notes on the State-of-the-Art. *Machine Learning 4*, 3/4 (December 1989), 377-394.

Brazdil, P.B. and Torgo, L. Knowledge Acquisition via Knowledge Integration. In *Current Trends in Knowledge Acquisition*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., IOS Press, May 1990, pp. 90-104.

Breuker, J. and Wielinga, B. Models of Expertise in Knowledge Acquisition. In *Topics in Expert System Design. Methodologies and Tools*. North-Holland, Guida, G. and Tasso, C., pp. 265 - 295, Amsterdam, 1989.

Brödner, P. Fabrik 2000, *Rainer Bohn Verlag* (1985).

Bullinger, H.J. and Salzer, C. Integration von CAD und CAP über ein gemeinsames Produktmodell. *CIM Management 6* ,(1989).

Bylander, T. and Johnson, T.R. Structured Matching: A Task-Specific Technique For Making Decisions. In *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, October 1-6 1989, pp. 8-1-8-19.

Chandrasekaran, B. Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. *IEEE Expert* (Fall 1986), 32 - 30.

Chase, W.G. and Ericcson, K.A. Skill and working memory. In *The Psychology of learning and motivation*. Academic Press, B., B., 1982.

Chi, M., Feltovich, P., and Glaser, R. Categorization and representation of physics problems by experts and novices. *Cognitive Science 5* , (1981), 121-152.

Clancey, W. Heuristic Classification. *Artificial Intelligence 27*(1985), 289 - 350.

Clancey, W.J. The Knowledge Level Reinterpreted: Modeling How Systems Interact. *Machine Learning 4*, 3/4 (December 1989), 285-291.

Diederich, J., Linster, M., Ruhmann, I., and Uthmann, T. A Methodology for Integrating Different Knowledge Elicitation Techniques. In *Proceedings of EKAW87*, Addis, T., Boose, J.H., and , Gains, B.R., University of Reading, 1987.

Eggen, J., Lundteigen, A.M., and Mehus, M. Integration of Knowledge from Different Knowledge Acquisition Tools. In *Current Trends in Knowledge Acquisition*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van S, M., IOS Press, May 1990, pp. 123-142.

Eversheim, W. Arbeitsvorbereitung. In *Organisation in der Produktionstechnik*. VDI Verlag, pp. 125-128, 1989.

Fandel G., Dyckhoff H. and Reese J. Industrielle Produktionsentwicklung, *Springer-Verlag* (1990)

Fischer, G., Lemke, A., and Mastaglio, T. Using Critics to Enpower Users. In *CHI 1990 Conference Proceedings*, 1990, pp. 337-347.

Fischer,U., Schmalhofer,F. Historische, Soziologische and Kognitive Aspekte der Fertigungstechnik. *Tech. Rept. Interner Bericht des ARC-TEC Projektes, German Research Center for Artificial Intelligence GmbH*, 1990.

Fox, J., Myers, C.D., Greaves, M.F., and Pegram, S. Knowledge Acquisition for Expert Systems: Experience in Lukemia Diagnosis. *Methods of Information in Medicine 24* (1985), 65-72.

Gaines, B.R. Design Requirements For Knowledge Support Systems. In *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, October 1-6 1989, pp. 12-1-12-20.

Hoffman,R. A Survey of Methods for Eliciting Knowledge of Experts. In: *SIGART Newsletter 108*,19-27, 1989.

Jacobson, C. and Freiling, M. Knowledge Dependencies. In *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, October 1-6 1989, pp. 16-1-16-19.

Jansen-Winkeln, R.M. Wastl: An Approach to Knowledge Acquisition in the Natural Language Domain. In *Proceedings of the European Knowledge Acquisition Workshop (EKAW'88)*, June 19-23 1988, pp. 22-1-22-15.

Karbach, W., Linster, M., and Voß, A. Models of Problem-Solving: One Label - One Idea?. In *Proceedings of EKAW90*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., University of Amsterdam, IOS Press , Amsterdam, February 1990, pp. 173 - 189.

Kidd, A. Human Factors in the Design and Use of Expert Systems. In *Fundamentals of Human-Computer Interaction*. New York Academic Press, Monk, A., 1985

Kintsch,W., Welsch,D., Schmalhofer,F., Zimny,S. Sentence Memory: A Theoretical Analysis. In *Journal of Memory and Language 29*, 133-159 (1990).

Kintsch,W. How Readers Construct Situational Models for Stories: The Role of Syntactic Cues and Causal Inferences. In A.F.Healy, S.Kosslyn, &R.M.Shiffrin (Eds.). *Essays in honor of William K.Estes Hillsdale, NJ:Erlbaum* (in press).

Kühn, O., Schmalhofer, F., and Schmidt, G. Diagnose von Wissenstypen für die Erstellung von Fertigungsplänen. *Tech. Rept. Interner Bericht des ARC-TEC Projektes, German Research Center for Artificial Intelligence GmbH*, 1990.

Lefkowitz, L.S. and Lesser, V.L. Knowledge Acquisition as Knowledge Assimilation. In *2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1987.

Legleitner,R. Überlegungen zum Produktmodell von ARC-TEC. ARC-TEC Diskussionspapier 90-02, *DFKI and Universität Kaiserslautern, FBK*, 1990.

Lewandowsky, St., Dunn,J. & Kirsner (eds) Implicit Memory: *Theoretical Issues*, Hillsdale, Lawrence Erlbaum, 1989

Lewis, C., Polson, P., Wharton, C., and Rieman, J. Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces. In *Human Factors in Computing Systems (CHI '90 Conference Proceedings)*, Chew, J.C. and Whiteside, J., Addison-Wesley, April 1-5 1990, pp. 235-242.

Linster, M. A Critical Look at KRITON: Moving from First to Second Generation Expert Systems. In *Proceedings of the 3rd Knowledge Acquisition Workshop*, Boose, J.H. and Gaines, B.R., Banff, 1988, pp. 17.1 - 17.19.

Martin, J. D. and Redmond M. Acquiring Knowledge by Explaining Observed Problem Solving. In *Knowledge Acquisition: SIGART Newsletter 108* (April 1990), 77 - 83.

Möller, J.U. Knowledge Acquisition from Texts. In *Proceedings of European Knowledge Acquisition Workshop (EKAW'88)*, 1988, pp. 25-1-25-16.

Mortik,K. Integration Issue in Knowledge Acquisition Systems. In *SIGART Newsletter 108*, 124-131, 1989.

Musen, M.A., Fagan, L.M., Combs, D.M., and Shortlife, E.H. Use of a domain-model to drive an interactive knowledge-editing tool. *International Journal of Man-Machine Studies* 26, (1987), 105 - 121.

Newell, A. The Knowledge Level. *Artificial Intelligence* 2(1982), 87 - 127.

Pfeiffer, J., Siepmann, T., and Teichmann, W. Expertensystem zur spanenden Bearbeitung. *Tech. Rept.* 2, *Technische Mitteillungen Krupp, Krupp GmbH*, 1988.

Richter, M., Boley, H., Wetter, T., and Warnecke, G. ARC-TEC: Acquisition, Representation and Compilation of Technical Knowledge. *Tech. Rept. Projektantrag, German Research Center for Artificial Intelligence, Postfach 2080,D-6750 Kaiserslautern*, 1989.

Richter,M.M. Expert Systems in mechanical engineering, In: *UNISYS Symposium Expert Systems- A Management Perspective, Saint Paul de Vence*, März 1990.

Riesbeck, C.K, Schank,R.C.. Inside case-based reasoning, *Lawrence Earlbaum* (1989).

Schmalhofer, F. and Wetter, T. Kognitive Modellierung: Menschliche Wissensrepräsentationen und Verarbeitungsstrategien. In *Künstliche Intelligenz*. Springer-Verlag, Christaller, T., Hein, H.W.W., and Richter, M.M., pp. 245-291, 1988.

Schmalhofer,F. and Glavanov, D. Three components of understanding a programmer's manual: Verbatim, propositional, and situational representations. In *Journal of Memory and Language 25*,279-294 (1986)

Schmalhofer,F. Expert Systems as Cognitive Tools for Human Decision Making. In: Mumpower,J.L., Lawrence,D.P., Renn,O., Uppuluri,V.R.R. (eds). *Expert Judgment and Expert Systems* 1987.

Schmidt, G., Legleitner, R., and Schmalhofer, F. Lautes Denken bei der Erstellung der Schnittaufteilung, der Werkzeugauswahl und Festlegung der Maschineneinstelldaten. *Tech. Rept. Interner Bericht des ARC-TEC Projektes, German Research Center for Artificial Intelligence GmbH*, 1990.

Schmidt, G. and Schmalhofer, F. Case-Oriented Knowledge Acquisition from Texts. In *Current Trends in Knowledge Acquisition*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., IOS Press, May 1990, pp. 302-312.

Schmidt, G. and Wetter, T. Towards Knowledge Acquisition in Natural Language Dialogue. In *Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW)*, 1989.

Simon,H.A. The Science of the Artificial. The *MIT Press*; Cambridge, MA; 1981.

SPK-Werkzeuge, F.A.P. Feldmühle Werkzeuge: Arbeitsbeispiele. 1989, Drehen und Fräsen.

Spur. Produktionstechnik im Wandel, Carl Hanser Verlag (1979).

Strube, G. and Janetzko, D. Episodisches Wissen und fallbasiertes Schließen: Aufgaben für Wissensdiagnostik und Wissenspsychologie. *Schweizerische Zeitschrift für Psychologie* (1990).

Swartout, W.R. XPLAIN: a System for Creating and Explaining Expert Consulting Programs. *Artificial Intelligence 21*(1983), 285-325.

Szpakowicz, S. Semi-Automatic Acquisition of Conceptual Structure from Technical Texts. *Workshop Banff* (November 1988), 35-1-35-16.

Thien-Chien-Wysk-85a An Introduction to Automated Process Planning Systems, Prentice-Hall Inc. (1985)

Thoben,J., Schmalhofer,F. Wiederholungs- Varianten- und Neuplanung bei der Fertigung rotationssymmetrischer Teile. *Tech. Rept. Interner Bericht des ARC-TEC Projektes, German Research Center for Artificial Intelligence GmbH*, 1990.

Weber, G. Cognitive diagnosis and episodic modelling in an intelligent LISP-tutor. In *Proceedings of the International Conference on Intelligent Tutoring Systems*, June 1988, pp. 207-214.

Wetter, T. First Order Logic Foundation of the KADS Conceptual Model. In *Current Trends in Knowledge Acquisition*, Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., IOS Press, May 1990, pp. 356-375.

Wielinga, B., Akkermans, H., Schreiber, G., and Balder, J. A Knowledge Acquisition Perspective on Knowledge-Level Models. In *Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop*, October 1-6 1989, pp. 36-1-36-22.

Woodward, B. Knowledge Acquisition at the Front End: Defining the Domain. *Knowledge Acquisition 2*, 1 (March 1990), 73 - 94.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

On Problems with the Knowledge Level Perspective*

Guus Schreiber * Hans Akkermans † Bob Wielinga *

*Department of Social Science Informatics, University of Amsterdam
Herengracht 196, NL-1016 BS Amsterdam, The Netherlands
Electronic mail: guussc/wielinga@swi.psy.uva.nl

†Netherlands Energy Research Foundation ECN
P.O. Box 1, NL-1755 ZG Petten, The Netherlands
Electronic mail: hans@ecn.UUCP

Abstract

In this paper some points of criticism on Newell's Knowledge Level Hypothesis are investigated. Among those are: the inability to represent control, the potential computational inadequacy, the lack of predictive power and the non-operational character (the problem of 'how to build it'). We discuss Sticklen's Knowledge Level Architecture Hypothesis in which he tries to overcome these problems. On the basis of general arguments as well as specific insights from our KADS knowledge level modelling approach we reject the points of criticism. We also argue that the extension Sticklen proposes is not necessary and partly also unwanted.

1 Introduction

The introduction by Newell of the *knowledge level hypothesis* has attracted considerable attention and stimulated new lines of research. It claims that there exists a distinct computer systems level that lies immediately above the symbol or program level. This knowledge level characterises the behaviour of problem solving agents in terms of their goals and actions, with knowledge serving as the medium, on the basis of a simple principle of rationality saying that an agent will carry out a certain action if it has knowledge that one of its goals can be achieved by that action [Newell, 1982].

It was Clancey who first showed the importance of this idea for the theory of knowledge-based reasoning [Clancey, 1983; Clancey, 1985b]. Since then, a variety of authors has elab-

*The research reported here was carried out in the course of the REFLECT project. This project is partially funded by the Esprit Basic Research Programme of the Commission of the European Communities as project number 3178. The partners in this project are the University of Amsterdam (Amsterdam, The Netherlands), the Netherlands Energy Research Foundation ECN (Petten, The Netherlands), the National German Research Centre for Computer Science GMD (St. Augustin, West-Germany) and BSR Consulting (Munich, West-Germany).

orated the viewpoint that the knowledge level is the right level of abstraction for knowledge acquisition and engineering [Wielinga et al., 1989; Bylander and Chandrasekaran, 1988; McDermott, 1989; Alexander et al., 1988; Musen et al., 1988; Steels, 1988]. This research has focused on the conceptual, implementation-independent aspects of knowledge and, on this basis, has contributed to generic models of problem solving and to more solid methodologies for KBS development.

Evidently, the knowledge level hypothesis has been a very fruitful one. On the other hand, it has also attracted strong criticism. This is well exemplified in a recent paper by Sticklen and the associated commentaries [Sticklen, 1989]. In brief, from discussions like these we single out as basic problems related to the knowledge level hypothesis:

1. *Computational inadequacy*: Due to the high level conceptual bias of knowledge level models, a very real danger is their potential computational inadequacy.
2. *Non-operational character*: Knowledge level models do describe knowledge that is necessary for problem solving, but give no clue as to how to build computational systems embodying and exploiting that knowledge.
3. *Inability to represent control*: Knowledge level models capture knowledge used in problem solving actions, but they do not provide ways to express the control of problem solving.
4. *Lack of predictive power*: Knowledge level models are useful to explain —from hindsight— the behaviour of certain problem solving agents (such as AI programs), but are unable to generate empirically testable predictions about that behaviour.

Interestingly, most of this criticism on the knowledge level hypotbesis appears to be inspired by what is seen by some as its greatest advantage: moving away from implementational issues in favour of the conceptual aspects of knowledge. In the present work, which is basically a *position paper*, the knowledge level hypothesis and its problems will be investigated in some detail. On the basis of general arguments as well as specific insights from our KADS knowledge level approach to KBS development [Wielinga et al., 1989], we will consider —and reject— the points of criticism raised above.

2 The Knowledge Level Hypothesis Debate

2.1 The Knowledge Level Hypothesis

The Knowledge Level Hypothesis (KLH) was put forward by Newell in his presidential address to AAAI-80 [Newell, 1982]. Newell discusses in this address the notions of *knowledge* and *representation* which is his view are central to AI. Newell signals that most of the work in AI is centred on representation. Representation refers here to the actual data structures and processes in an (AI) program. He suggests that the confusing issues in AI research on representation may (partly) be due to the limited attention the research community is giving to the study of the nature of knowledge. The Knowledge Level Hypothesis is

aimed at providing a platform for studying knowledge independent of its representation in a programming language. Newell phrases the KLH as follows:

> "There exists a distinct computer systems level, lying immediately above the symbol level, which is characterised by knowledge as the medium and the principle of rationality as the law of behaviour."

Representations (data structures and processes) are part of the the symbol level, whereas knowledge is the prime ingredient (the medium) at the knowledge level. Newell sketches the structure of the knowledge level as an agent that has a physical body (consisting of a set of actions), a body of knowledge, and a set of goals (goals are bodies of knowledge about the state of the environment). The principle of rationality that governs the behaviour of the agent is formulated as follows:

> If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select this action.

The central issue now becomes: what is the nature of the knowledge the agent has? Newell characterises knowledge as a"competence-like notion, being a potential for generating action" and as "entirely *functionally* in terms of what it does, not *structurally* in terms of physical objects". A representation is defined as a "symbol system that encodes a body of knowledge".

As an example of the usefulness of a distinction between knowledge level and symbol level, Newell points to the work of Schank on conceptual dependency structures [Schank, 1975]. He argues that the main contribution of this work is at the knowledge level – namely by providing a quite general way of describing knowledge of the world. Although AI researchers felt that this work was incomplete without an implementation, the actual program added little to the theoretical work: it was just a large AI program with the usual ad hoc constructions.

2.2 Knowledge Level Modelling in Knowledge Acquisition

One of the areas where the knowledge level hypothesis has received considerable attention is the field of knowledge acquisition. Experience with the first generation of knowledge-based systems (MYCIN and its derivatives) showed that the *transfer approach* to knowledge acquisition was simply inadequate. In the transfer approach the knowledge engineer tries to extract knowledge from a domain expert in the form of the representation in the system (e.g. as production rules). The problems with this approach are manifold: the mapping from elicited expertise-data onto the required representation is difficult and often not possible; systems with a large knowledge base become difficult to maintain; explanation facilities are poor; etc. The main reason for this is that the gap between the observed problem solving behaviour and the target application is just too wide. What is needed is an intermediate description of the expertise in a task domain at a more abstract level. The knowledge level provides precisely this intermediate level of description.

Broadly speaking, two approaches can be identified to the use of the idea of a knowledge level description in knowledge acquisition, In the first approach the starting point is an implementation of (parts of) a problem solver. Here, knowledge level notions are introduced by providing abstract, implementation-free, descriptions of the knowledge elements required by the problem solver. Examples of this approach are the Generic Task approach [Bylander and Chandrasekaran, 1988] and the work of McDermott et al. on MOLE, SALT and other systems [McDermott, 1989]. In the second approach, exemplified in the KADS methodology [Wielinga et al., 1990], the knowledge level descriptions are part of a conceptual model of a task domain. The conceptual model serves as a specification of the (knowledge) requirements for a particular knowledge-based application. The conceptual model is not directly linked with the actual implementation.

2.3 Criticism of the Knowledge Level Hypothesis

The introduction of the KLH has led to criticism both of the KLH itself and also of the application of knowledge level descriptions in the development of knowledge-based systems. In a recent article, Sticklen [Sticklen, 1989] phrases these points of criticism in the form of an extension of the Knowledge Level Hypothesis: the "Knowledge Level Architecture Hypothesis". We will go through his argument in some detail, as his discussion is in a sense typical for critics of the KLH.

First, Sticklen acknowledges that the notion of a knowledge level description is a useful one. He points to the work of Clancey on Heuristic Classification [Clancey, 1985b] as a prototypical example of a knowledge level description of a problem solving agent (process). In his article, Sticklen apparently views the "horse shoe" inference structure of heuristic classification as a complete knowledge level description. He then concludes that this type of description is unable to yield verifiable predictions of the problem solving behaviour of an agent and that thus the knowledge level hypothesis is incomplete. This conclusion is based on the assumption that a scientific theory has two necessary components: (i) the theory must account for known phenomena and (ii) the theory must make (verifiable) predictions about phenomena that will be observed in the future. He argues that the lack of predictive power of knowledge level descriptions is due the fact that there is no way in Newell's knowledge level to specify problem solving control. In his Knowledge Level Architecture Hypothesis he proposes to extend the KLH with the possibility of decomposing an agent (task) into sub-agents (sub-tasks) and allow specification of ordering among sub-agents. This extension seems harmless enough. Newell himself is not very clear whether decomposition of agents is allowed. A requirement for the decomposition is that the resulting sub-agents are 'knowledgeable', i.e. that they can be described in knowledge level terms. Sticklen is however not very clear on how this can be ensured.

Given a decomposition of an agent into sub-agents, Sticklen defines the corresponding knowledge level architecture through two ingredients: (i) a specification of the communication paths between sub-agents, and (ii) the specification of the message protocols for inter-agent communication.

Sticklen claims that a knowledge level architecture description provides a

"*blueprint* for how to build a problem solver that may be used as a simulator."[1]

The simulator provides the required predictive part of the theory, The role of the simulator can be compared with the role of numeric simulations in physics.

In his article, Sticklen tries to cope with some –highly interrelated– points of criticism of the knowledge level approach,. He mentions two points explicitly: the *inability to represent control* and the *lack of predictive power*. Two other points of criticism are:

computational inadequacy:
When a knowledge level description is transformed into a symbol level description, what kind of guarantees does one to have that the resulting system is computationally adequate?

non-operational character:
A knowledge level model gives no clues about how to build an operational system. I.e. it does not solve the 'design problem'.

These last two points are also covered by the knowledge level architecture hypothesis through the blueprint for building a simulator (i.e. a knowledge-based application).

In the rest of this article we discuss the various points of criticism. We will argue that the extension that Sticklen proposes is not necessary and partly also unwanted.

3 Representing Control in Knowledge Level Models

Sticklen's Knowledge Level Architecture Hypothesis encompasses an extension of the knowledge level with respect to the description of problem solving control. In our view knowledge about task (agent) decompositions and dependencies is indeed just another type of knowledge with its own specific characteristics that should —and can– be described in a knowledge level model. The description in [Clancey, 1985a] of the diagnostic strategy developed for NEOMYCIN is a good example of what we would call a knowledge level description of problem solving control. Clancey calls this a "competence model" of diagnostic strategy, which already indicates that the description has the knowledge level flavour Newell is aiming at in his KLH.

In our opinion, the question whether this type of (competence-like) control knowledge is or is not present in the Knowledge Level Hypothesis, as originally stated by Newell, is not very important or interesting. From our experience it is clear that the above-mentioned type of control knowledge is a necessary and important ingredient of a practical knowledge level theory of problem solving. In addition to the task decompositions and inter-dependencies (the task knowledge in the KADS model, see [Wielinga et al., 1989]) we would also add

[1] italicisation by the authors

meta-knowledge about a problem solving agent as a separate kind of control knowledge (somewhat confusingly termed 'strategic knowledge' in KADS). This strategic or tactical knowledge is an important ingredient for building more flexible knowledge-based systems.

The problem of ensuring that the decomposition results in 'knowledgeable' sub-agents can be handled by providing a knowledge level typology of canonical inferences (the lowest level of sub-agents), thereby ensuring that such agents can indeed be described in knowledge level terms. Examples of canonical inferences are the three steps in the Heuristic Classification inference structure (abstraction, association and refinement) [Clancey, 1985a] and the set of knowledge sources defined in KADS [Breuker and Wielinga, 1989]. Clearly, much work still needs to be done in this area to arrive at a coherent and more or less complete typology.

What worries us in the Knowledge Level Architecture idea is that it should provide a blueprint for building a simulator. Whereas we think that it is very well possible to define structured ways of building a knowledge-based system from a knowledge-level specification (see Sec. 6), we do not believe, that a knowledge level model should –and can– contain all information necessary for building the implementation. In fact, by nature it should not! In the process of implementing a system it will always be necessary to add specific information. The design process is constrained by the knowledge level specification, but a large number of design decisions concern issues that are not relevant at the knowledge level.

We think that the confusion arises from the way in which the word "control" is used. There is a difference between what we would call respectively *knowledge level control* and *symbol level control*. The description of the diagnostic strategy of NEOMYCIN in [Clancey, 1985a] is a clear example of knowledge level control. This type of knowledge concerns task decompositions of and orderings between knowledgeable agents and possibly also meta-knowledge about agents. Symbol level control is concerned with the control issues that arise when a particular representation or AI technique is selected to realise a problem solving agent. A similar distinction between these two types of control is made by Gruber [Gruber, 1989; page 5].

Although symbol level control is not an issue at the knowledge level, it does pose several problems that have to have solved in the implementation of a particular application. One only has to look at the vast amount of ' 'symbol level" AI research to conclude that these problems are by no means trivial. Thus, specification of knowledge level control does *not* provide a blueprint for building a simulator.

4 Epistemological and Computational Adequacy

Knowledge level models give a high level description of the knowledge as it is utilised in problem-solving reasoning. Although terminology is different, a common view appears to be emerging in the literature based on the idea that the knowledge level is constituted by different types and components of knowledge, and that these forms of knowledge play different roles in the reasoning process and have inherently different structuring principles.

In addition, the knowledge level approach attempts to demonstrate that many of these knowledge types and components have a generic character, *i.e.*, they are applicable to a broad class of tasks and/or domains. In this sense, generic knowledge components can be viewed as intermediate in the continuum from weak to strong methods.

The use of knowledge level models lies in yielding a high level and intuitive *explanation of reasoning behaviour*. Moreover, generic models at the knowledge level are important vehicles for knowledge acquisition, because they provide the knowledge engineer with *reusable interpretation 'templates'* that guide the analysis and organisation of elicitation data.

As a consequence of the high-level conceptual nature, a major objection to the use of knowledge level models in the KBS development process is their potential computational inadequacy. Since knowledge level models do not specify the operational control regime in full detail they are apt to potential combinatorial explosion behaviour. The generic task approach by Chandrasekaran *et al.* takes the view that the knowledge level description and the operational problem-solving method as employed in the computational system cannot be separated. In contrast, the choice of computational techniques to realise a certain knowledge level function is seen in KADS as part of the design activity in knowledge engineering (further discussed in Sec. 6). An important criticism of the KADS-type of conceptual models is therefore that —although they are very useful from a practical epistemological viewpoint— they do not guarantee computational tractability.

4.1 The knowledge level: role limitations

In this very broad and general way of speaking, the criticism above is correct: knowledge level models do not make statements about computational adequacy as such. However, we claim that the *structure* of knowledge level models as outlined previously provides important safeguards against the computational inadequacy. The combinatorial complexity at the computational level is caused in our view by the unrestricted applicability of and access to knowledge as present in the knowledge base. The underlying principle of knowledge level modelling is *imposing structure through knowledge differentiation*. This is achieved by distinguishing within the involved body of knowledge different types and components that play specialised roles in the totality of the problem-solving process.

An example is the cover-and-differentiate method [Eshelman et al., 1988] for diagnosis which takes the following steps:

1. Determine events (hypotheses) that potentially explain symptoms.
2. Identify information that can differentiate between candidate explanations by ruling out, providing support for candidates, providing preferences.
3. Get this differentiating information and apply it.
4. If new symptoms become available, go to step 1.

In this example, several types of knowledge are specified in an informal way. First, the domain should provide concepts like: event, symptom, explanation link, preference, rule-out relations etc. Second, a number of basic inference types (similar to KADS knowledge sources [Breuker and Wielinga, 1989]) are defined: generating a hypothesis, matching a hypothesis to the available data, selection or ordering, compute preferences. In addition there is control knowledge that indicates that all possible candidates are generated given a set of symptoms, and that differentiating information is obtained in a backward manner. These aspects of knowledge would be categorised as task knowledge in KADS.

The differentiating requirements and constraints as laid down in the knowledge model express the specific role that the considered knowledge plays in the problem-solving process. This limits the use that can be made of that knowledge. For instance, a logical implication sentence of a certain type can be specified to be usable only for matching inferences, and not for other types of inference steps (like, say, a selection). KADS notions such as knowledge sources, metaclasses and task structures [Breuker and Wielinga, 1989] generally yield the possibility of selecting specific rules or theories needed to produce a certain inference. In this way the knowledge level model provides *role-limiting constraints* [McDermott, 1989] to the use of knowledge.

4.2 The computational level: access limitations

Thus, an essential epistemological feature of knowledge level models is that they specify role limitations of knowledge. The corresponding notion at the computational level is that of access limitations[2]. The knowledge that is specified in a KADS conceptual model cannot be used in arbitrary ways: it has to fulfill certain typing requirements and can only be applied in accordance with the constraints specified by the model. Given that knowledge components can only be selectively used due to specified role-limiting constraints, the consequence will be a restricted access to the computational representation of those components in a knowledge-based system. Assuming that (as said) the computational complexity results from unrestricted applicability of and access to knowledge, the structure of a knowledge model has a profound effect on the computational adequacy. If the knowledge model sufficiently refines the various knowledge types and explicitly indicates what inferences access what domain structures, the computational complexity will be greatly reduced.

Our position is supported by experiences from AI. Here, a common method to achieve computational tractability is to introduce structural differentiations that at the knowledge level can be characterised as adding new role specialisations and limitations of knowledge. As a matter of fact, this attitude is quite clearly exemplified in the work on generic tasks. Also heuristic classification is a good specimen: if direct association between data and solutions results in a computationally inadequate model, a possible method to obtain a more tractable model is the introduction of additional inference steps, *viz.*, abstraction and refinement. The associated knowledge can now be specified to be accessible only in a restricted part of the 'horse-shoe' inference structure, whereas this is impossible in a simple direct-association model. Yet another interesting example is provided by the work of Patil

[2]Cf. Newell's slogan: *representation = knowledge + access*

on medical diagnosis [Patil, 1988], showing that complicated forms of diagnosis can be gradually built up by starting from a simple generate-and-test method and subsequently introducing new elements of knowledge differentiation, so as to preserve the computational adequacy with increasing task complexity.

In conclusion, we suggest that *epistemological role limitations as described by a knowledge level model are connected to computational access limitations.* Computational adequacy cannot be strictly guaranteed, but the knowledge level approach does provide significant handles on the computational tractability by means of the role and access limitations ensuing from knowledge differentiation.

5 Do Knowledge Level Models Yield Predictions?

Sticklen puts forward as a central objection to the knowledge level hypothesis in its current form its lack of predictive power. In his view, knowledge level models are capable of explanatory analysis of the reasoning behaviour of intelligent systems in retrospect, but the do not generate empirically verifiable predictions. He discusses this in the context of the broad question whether AI can be cousidered to be a science when it does not contain a predictive component.

First of all, Sticklen's equation of science with predictive power needs some qualification. On this score, we basically agree with the critical commentaries on his position. Restricting ourselves to physics —Sticklen's favourite example of an 'established science'— it is clear that explanatory and predictive power constitute much more subtle ingredients of science than he suggests. Especially in branches of physics that are close to engineering we encounter phenomenological models that have strong predictive power but no explanatory status. This is the case if such models numerically express important regularities of experimental data without referring to more fundamental laws of physics (this may be achieved by, simply speaking, a linear regression analysis of a certain large amount of data that happens to be successful in the general case). Conversely, there are models that are based upon basic physical principles (and so are felt to be relevant in providing a physically intuitive picture of a process) but refer to phenomena that are far beyond any experimental test (such as processes on time scales that are orders-of-magnitude shorter than can be measured)[3]. In general, physical theory is an iceberg of which only a tiny part is visible for empirical verification. An important aspect is that physical theories only yield predictions at a certain level and within a certain regime. For instance, macro-physical theories such as hydrodynamics do make predictions about, say, fluid pressure and waves, but not about microphysical entities like atomic interaction potentials within water molecules. The latter notions have even no existence at the macro-physical level, whereas in the microphysical theory there is no place for concepts like fluid, waves and pressure. Thus, each level of physical description generates its own type of predictions.

[3]Our examples are based upon personal experience in mainstream nuclear physics and engineering. If we accept as a simple operational definition of what may count as science: the publication of work in refereed international journals on a regular basis (the latter to rule out incidental mistakes of reviewers), all these examples must be accepted as science. We can provide the interested reader with pertinent references to the literature.

This parallel carries over to modelling for knowledge-based systems. A knowledge level model describes what types of reasoning steps an agent that is being modelled is expected to take in performing a certain task. In addition it makes certain claims about the structure of the domain-specific knowledge involved, and it specifies the strategic elements of the reasoning. For example, the cover-and-differentiate knowledge model outlined earlier tells us that certain hypotheses will be discarded in a *rule-out* inference step. How such an inference will operationally manifest itself depends on the system under investigation. An AI program may print out the removal of a hypothesis from the list of current candidates, while a human expert may utter a natural language sentence implying that the hypothesis is no longer considered. Although the operational form may differ, in both cases the ruling-out of a hypothesis is in principle empirically verifiable. Similarly, the heuristic-classification model would predict that much of the reasoning effort would be concentrated in data reduction and not in hypothesis handling, as is the case in the cover-and-differentiate method.

Thus, a knowledge level model may he applied to both human and artificial problem solvers. In either case, however, it is hard to see why knowledge models as we have sketched them possess no predictive power, as seems to be Sticklen's complaint. But it has to be acknowledged that the corresponding predictions are of a certain kind only, namely, on the level of the reasoning steps —which type, under what knowledge conditions— that we expect an agent to perform. No predictions are made concerning the detailed, operational or symbol, level of observation. As corroborated by much of the literature concerning the philosophy of science, testing a theory requires an interpretation of the observations that we make concerning the real world. Evidently, the regime of prediction (and, thus, of validity) of knowledge level models is limited — in this case to the type of and the conditions for inference steps to be carried out by an intelligent agent. Nevertheless, as pointed out, a limited regime or level at which predictions can be made is standard not only in AI but also in physics. Consequently, we disagree with Sticklen's criticism on the predictive power of knowledge level models.

6 System Building on the Basis of Knowledge Level Models

Knowledge level models do not contain all information necessary for the implementation of a system. In the KADS approach to developing knowledge-based systems, a separate *design model* is introduced [Schreiber et al., 1988]. In this design model appropriate AI techniques and representations are selected to realise the problem solving behaviour specified in the knowledge level model. The design model is thus a specification of the symbol level notions such as data structures and processes. The design model is also the place where additional, *symbol level*, control (cf. Sec. 3) is specified.

Fig. 1 provides a graphical representation of the different roles that the knowledge level model and the design model in our view play in the development process of a knowledge-based system. The knowledge level model is constructed by an observer of problem solving phenomena (e.g. human expertise). The observer (knowledge analyst) is aided in this task

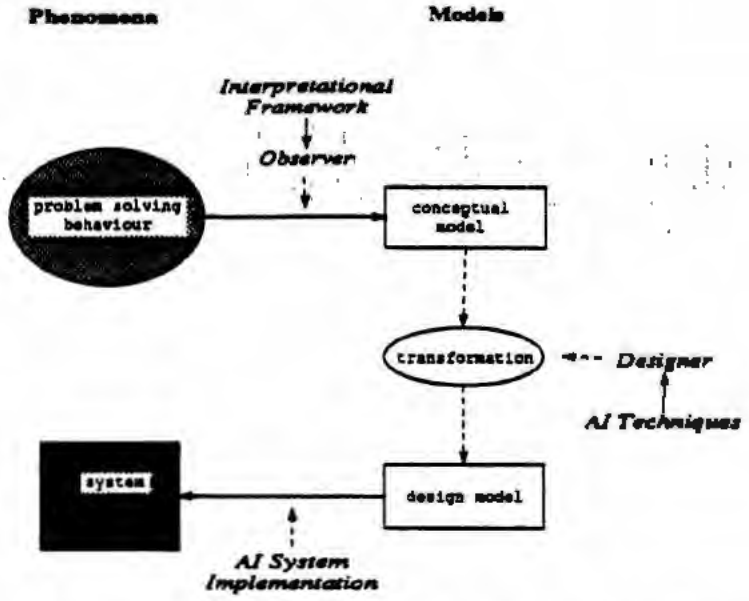


Figure 1: Role of the conceptual (knowledge level) model and of the design (symbol level) model in the development of knowledge-based systems

by an interpretational framework, that should consist of two parts: (i) a vocabulary for describing various knowledge types, such as the categorisations in the KADS conceptual model, and (ii) generic knowledge components. These generic components are partial instantiations of knowledge reoccurring in a class of tasks and/or domains. Heuristic classification, cover-and-differentiate and the interpretation models in KADS [Breuker et al., 1987] are examples of such generic components.

The design model is constructed by transforming the knowledge-level descriptions into symbol-level descriptions through the selection of appropriate AI techniques and representations, that realise the specified problem solving behaviour. The design model provides the basis for implementing the physical system. Although the designer is in principle free to develop any design model that meets the requirements of the knowledge level model, we are strongly in favour of a *structure-preserving* design. With this we mean that there is a structural correspondence between knowledge level elements and symbol level elements. This approach is in line with Pylyshyn's "cognitive penetrability" requirement for programs simulating cognitive behaviour [Pylyshyn, 1986]. Structural correspondence paves also the way for explaining the computation of a program at various levels of abstraction (physical, symbol/functional, knowledge). To support this structure-preserving design we have developed a *design description language* [Schreiber et al., 1989]. This language is a notational device that enables a design in which the transformation is basically a process of *adding* symbol level information, such as operational control, to the knowledge level model. We are currently embarking on an ESPRIT-II endeavour in which this approach will be explored further, for example by providing a catalogue of AI techniques and representations together with corresponding shellifications. Given the state of the art, such a catalogue will however for the time being remain incomplete.

7 Conclusions

In this paper we have investigated some points of criticism on Newell's Knowledge Level Hypothesis: the inability to represent control, the potential computational inadequacy, the lack of predictive power and the non-operational character (the problem of 'how to build it').

We have pointed at a possible confusion between two types of control. We have indicated that in our view there is a need to model in a practical knowledge theory of problem solving a particular type of control knowledge with a strong knowledge level flavour as for example decompositions of knowledgeable agents and meta-knowledge about agents. If this type of knowledge level control was not part of Newell's original hypothesis, we feel that it should be extended in this respect. We disagree with an extension such as Sticklen's Knowledge Level Architecture Hypothesis, as it requires a type of control specification that inherently belongs to the symbol level. We have argued that the role-limitations in knowledge level descriptions give rise to access limitations at the symbol level. Together they provide important safe-guards against potential computational inadequacy. We have also explained that knowledge level models as we propose them allow limited forms of prediction – namely at the level of the type of and the conditions for inference steps carried out by an intelligent agent. We have shown that this is not different from the role of formal theories in predicting physical phenomena. Finally, we have argued that in the process of system building there is a place for a separate design model. We view design as a process of adding symbol level information to a knowledge level model. Notational devices, catalogues, and shellifications can support the design and implementation, but by nature these activities are open-ended, i.e. the solution space is large.

Although we feel that the original Knowledge Level Hypothesis still stands, much work still needs to be done to 'make it work': to arrive at a practical knowledge level theory of problem solving. Looking at the various approaches to knowledge level modelling it is clear that the terminology used to describe the knowledge level is confusing and ambiguous. In our opinion there is a clear need for a *formal* framework for describing knowledge level models. We fully agree with Newell [Newell, 1982; p. 121-122] that the major role of logic in AI should be to support the analysis of the knowledge level. In [Akkermans et al., 1990] a first effort is made to devise a logical framework for knowledge level analysis. Although this work can only be seen as a first start, we feel that this is one important research direction for improving the state of the art in knowledge acquisition.

Acknowledgement Frank van Harmelen provided useful comments on an earlier version of this paper.

References

Akkermans, H., van Harmelen, F., Schreiber, G., and Wielinga, B. (1990). A formalisation of knowledge-level models for knowledge acquistion. *International Journal of*

Intelligent Systems. Forthcoming.

Alexander, J., Freiling, M., Shulman, S., Rehfuss, S., and Messick, S. (1988). Ontological analysis: an ongoing experiment. In Boose, J. and Gaines, B., editors, *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 25–37. Academic Press, London.

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billault, J.-P., Davoodi, M., and Hayward, S. (1987). Model Driven Knowledge Acquisition: Interpretation Models. ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd,.

Breuker, J. A. and Wielinga, B. J. (1989). Model Driven Knowledge Acquisition. In Guida, P. and Tasso, G., editors, *Topics in the Design of Expert Systems*, pages 265–296, Amsterdam. North Holland.

Bylander, T. and Chandrasekaran, B. (1988). Generic tasks in knowledge-based reasoning: The 'right' level of abstraction for knowledge acquisition. In Gaines, B. and J.Boose, editors, *Knowledge Acquisition for Knowledge Based Systems*, volume 1, pages 65–77. Academic Press, London.

Clancey, W. (1983). The epistemology of a rule based system -a framework for explanation. *Artificial Intelligence*, 20:215–251. Also: Stanford Heuristic Programming Project, Memo HPP-81-17, November 1981, also numbered STAN_CS-81-896.

Clancey, W. (1985a). Acquiring, representing and evaluating a competence model of diagnostic strategy. In Chi, Glaser, and Far, editors, *Contributions to the Nature of Expertise*.

Clancey, W. (1985b). Heuristic classification. *Artificial Intelligence*, 27:289–350.

Eshelman, L., Ehret, D., McDermott, J., and Tan, M. (1988). MOLE: a tenacious knowledge acquisition tool. In Boose, J. and Gaines, B., editors, *Knowledge Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 95–108, London. Academic Press.

Gruber, T. (1989). *The Acquisition of Strategic Knowledge*. Perspectives in Artificial Intelligence, Volume 4. Academic Press, San Diego.

McDermott, J. (1989). Preliminary steps towards a taxonomy of problem-solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255. Kluwer Academic Publishers, The Netherlands.

Musen, M., Fagan, L., Combs, D., and Shortliffe, E. (1988). Use of a domain model to drive an interactive knowledge editing tool. In Boose, J. and Gaines, B., editors, *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 257–273, London. Academic Press.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 1982:87–127.

Patil, R. (1988). Artificial intelligence techniques for diagnostic reasoning in medicine. In Shobe, H. and AAAI, editors, *Exploring Artificial Intelligence: Survey Talks from the National Conferences on Artificial Intelligence*, pages 347–379. Morgan Kaufmann, San Mateo, California.

Pylyshyn, Z. (1986). *Computation and Cognition: Toward a Foundation for Cognitive Science*. MIT Press, Cambridge, USA.

Schank, R. (1975). *Conceptual information processing*. North Holland, Amsterdam.

Schreiber, G., Breuker, J., Bredeweg, B., and Wielinga, B. (1988). Modelling in KBS development. In *Proc. 2th European Knowledge Acquisition Workshop, Bonn*, pages 7.1– 7.15, St. Augustin. GMD. GMD-Studien 143.

Schreiber, G., Wielinga, B., Hesketh, P., and Lewis, A. (1989). A KADS design description language. ESPRIT Project 1098, deliverable B7 UvA-B7-PR-007, University of Amsterdam.

Steels, L. (1988). Components of expertise. AI Memo 88-16, AI Lab, Vrije Universiteit Brussel. Published in *AI Magazine*, summer 1990.

Sticklen, J. (1989). Problem solving architecture at the knowledge level. *Journal of Experimental and Theoretical Artificial Intelligence.*

Wielinga, B., Akkermans, H., Schreiber, G., and Balder, J. (1989). A knowledge acquisition perspective on knowledge-level models. In Boose, J. H. and Gaines, B. R., editors, *Proceedings Knowledge Acquisition Workshop KAW'89, Banff*, pages 36–1 – 36–22, University of Calgary. SRDG Publications.

Wielinga, B., Boose, J., Gaines, B., Schreiber, A., and van Someren, M., editors (1990). *Current Trends in Knowledge Acquisition.* IOS, Amsterdam.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Acquiring Knowledge for Validation Models

Evangelos Simoudis*
Digital Equipment Corporation
AI Applications Group

James Miller
Digital Equipment Corporation
Cambridge Research Laboratory

Abstract

We present a methodology for building validation models, the knowledge structure used during the retrieval phase of our case-based reasoning systems. This methodology allows a knowledge engineer to start with a textual database of previously solved cases and quickly produce an effective case-based reasoning system. Using our methodology, we have built two such systems; one of these systems operates on a domain spanning 60% of the problems presented to an existing rule-based expert system. By comparison, the rule-based system took four times as long to construct at an estimated cost six times as great. While our system successfully solves 100% of the problems in its domain, the rule-based system is only capable of solving 40% of the problems in the full domain.

*The authors wish to thank David Waltz for his help with this research.

1 Introduction

We are studying retrieval in the context of case-based problem-solving. Our goal is to utilize existing databases as case bases, retrieving all the cases that are relevant to a new problem, while retrieving as few as possible of the irrelevant ones. We employ a two-step retrieval process: retrieval based on surface features (SFB retrieval), followed by validation of the retrieved cases. The first step uses easily acquired information about surface features, but it is not sufficiently discriminating. In our studies, 11% and 7.6% of the cases in databases were returned in each SFB retrieval. The second step, validation, uses knowledge about tests that can be performed on the new problem. We have proposed[13] encoding this knowledge in a validation model. The combination of SFB retrieval and validation reduces the number of cases retrieved to 1.5% and 0.58% respectively, without omitting any relevant cases.

In this paper, we present a methodology for building validation models. This methodology allows a knowledge engineer to start with a textual database of previously solved cases and quickly produce an effective case-based reasoning system. Using our methodology, we have built two such systems. The first of these systems, CASCADE, operates on a domain spanning 60% of the problems handled by an existing rule-based expert system, CANASTA. By comparison, CANASTA took four times as long to construct as CASCADE at an estimated cost six times as great. While our CASCADE system successfully solves 100% of the cases in its domain, CANASTA is capable of only solving 40% of the cases in the full domain.

Our methodology reduces the cost of building the system by allowing the knowledge engineer to begin the knowledge acquisition task without contacting a domain expert; instead, the engineer reads the existing database. Based on this, and other background reading, the engineer prepares a proposed validation model. The engineer and the expert then meet (for the first time) to begin the iterative process of refining the validation model. The process terminates when the expert agrees that the model is correct and the engineer agrees that it has sufficient detail to encode the cases from the original database. Finally, the engineer encodes each case from the database using the validation model, producing a working expert system.

2 Related Work

A number of other case-based reasoning systems use retrieval based on surface features followed by a justification phase. These systems use the most common form of deeper knowledge, causal models, for reasoning during justification. While CASEY[8] uses a pre-existing causal model, CHEF[6], PRIAR[7], KRITIK[5], CYCLOPS[14], and PROTOS[2] all develop their own causal models for their domains. None of these systems addresses the knowledge acquisition problem involved in building a causal model. PROTOS, however, uses a causal model to facilitate the acquisition of classification knowledge. Classification knowledge is more simply structured than causal knowledge, and is thus easier to acquire. In fact, we suspect that the designers of PROTOS would agree with us: it is hard to acquire the knowledge needed to build a causal model. Unlike these systems, we entirely avoid the problem of building a causal model. Instead, we propose the acquisition of validation models; these models are much simpler to construct than causal models (but still considerably more difficult than the simple classification knowledge of PROTOS).

PRIAR and KRITIK use causal explanations during case reasoning to adapt existing solutions to fit new problems — an important area, but one that our work does not explore. CYCLOPS concentrates on the reverse of our problem: we are concerned with retrieving too many cases for careful scrutiny, while in CYCLOPS the concern is that no relevant cases may be retrieved. CYCLOPS uses deep knowledge to modify the indices used during the retrieval phase, an approach that meshes nicely with ours to produce an automated method for improving the performance of SFB retrieval.

In addition to the work of the case-based reasoning community, our work is related to that of the knowledge acquisition community. Tools for the automatic acquisition of knowledge have concentrated on specific types of knowledge (for example, Bennett's ROGET[3] concentrates on application domain conceptual structure and Minton's PRODIGY[12] concentrates on control knowledge). In addition, Davis's TEIRESIAS[4], Anderson's GEOMETRY TUTOR[1], Newell's SOAR[9], and Mitchell's LEAP[11] refine and compile (rather than acquire) existing knowledge. Our work, however, attempts to improve the overall task of the knowledge engineer. In fact, we do *not* provide an automation tool, but rather define a more efficient methodology which we hope will subsequently be aided by automated tools.

3 Acquiring Knowledge: Our Viewpoint

We are particularly concerned with the engineering aspects of knowledge-based systems: building an efficient system in a reasonable amount of time and at an acceptable cost. At present, the cost of creating production-quality systems is high and they take too long to construct. Despite the proliferation of rule-based expert systems, implementing each such system remains very much an art that is performed by a knowledge engineer in conjunction with a domain expert. As with other programs, rule-based systems are dominated by a large number of components designed to handle special situations, and are plagued with interdependencies between these and other components.

We consider two problems, *time* to build and *cost* of building, separately; the dominant factor is different in the two cases and the solutions may well be different. The time to build an expert system is dominated by the efficiency of the knowledge engineer, and can be improved by such things as expert system shells. The cost, however, is often dominated by the amount of time required of a domain expert. These experts are hard to locate, their time is often overcommitted to priority projects elsewhere, and they are very highly paid. A reasonable estimate is that a domain expert costs eight to ten times as much, per day, as a knowledge engineer when all of the ancillary costs are considered.

Fortunately, knowledge acquisition is at the heart of *both* of these problems. Our methodology provides a specific knowledge representation scheme (the validation model), thus removing one major design decision from the knowledge engineer's concern, and consequently increase productivity. We significantly reduce the costs by using an existing database of cases (prepared by domain specialists[1] and experts as part of their routine work) to allow the knowledge engineer to acquire domain knowledge without direct assistance from an expert. Furthermore, the knowledge engineer's use of the database enables knowledge acquisition sessions to be very carefully focussed, again reducing

[1] In both of our applications we have identified individuals who have specialized knowledge of the domain but cannot be classified as experts. In general, these people have operational knowledge of the area, but don't necessarily understand complex interactions or causality within the domain. For example, most general practitioners of medicine would qualify as domain specialists: they have sufficient general knowledge to diagnose common problems, but they lack the knowledge needed to handle rare or specialized problems. We find it useful to distinguish specialists from experts because of the difference in impact on the cost equation: specialists may be costly, but experts are downright expensive.

the expert's time and consequent costs.

4 A Tale of Two Systems

This section compares the time and cost of developing two different expert systems, both for diagnosing crashes of the VMS operating system. The first system, CANASTA, is rule-based and was developed by another group within Digital; it attempts to handle all forms of crashes. We developed the second system, CASCADE, concurrently but independently, using the methodology described in this paper. It is case-based, and is designed to handle only crashes related to device driver failures, which account for 60% of the problems encountered in practice. Section 6 presents additional details of CASCADE, as well as a second case-based system we have developed.

Figure 1 compares CANASTA to CASCADE, showing a breakdown of the

	CANASTA	CASCADE
Method	265 rules	200 cases
Domain	VMS crashes	VMS crashes
Subdomain	All	Device-Driver[1]
Success Rate	40%	60%
Acquisition time (pdays)	80[2]	20[3]
Encoding time (pdays)	600	85[4]
Maintenance time (pdays)[5]	280	0
Total time (pdays)	960	105
Total cost (est.)[6]	1600	205

1. Crashes related to device-drivers account for 60% of all crashes reported for VMS.
2. Spent with a domain expert.
3. 5 pdays spent with a domain expert and 10 pdays with a domain specialist.
4. This includes one-time development that would not be repeated for another expert system using the same methodology.
5. See the text for a definition of maintenance time.
6. This estimate is based on a domain expert costing eight times a knowledge engineer and a domain specialist as four times a knowledge engineer.

Figure 1: Two Expert Systems: CANASTA and CASCADE

overall development cost into three disjoint components: knowledge acquisition, knowledge encoding, and maintaining the knowledge. By "maintaining the knowledge" we mean, precisely, the time required to merge two self-consistent sets of rules (an initial working set of rules and a set of rules required to encode newly acquired knowledge) to produce a self-consistent whole. Notice that our definition of maintenance is *not* the traditional distinction between time spent prior to shipping an initial version versus that spent later; rather we are apportioning all of the time spent (before or after shipping) into one of three disjoint tasks. Figure 1 shows that the cost of developing CANASTA, the rule-based system, was 7.8 times higher than for CASCADE, and the development time was over 9 times as long.

These development times need to be examined carefully, however. There are two contradictory factors that must be considered. On one side, the savings may be even larger than these figures indicate. Since CASCADE was the first system we developed using our methodology, part of the 85 pdays spent encoding knowledge includes the initial design of the data structures used for the validation model. This design time is not repeated for later systems, since the same basic data structure are used for all validation models[2].

On the other hand, it isn't quite fair to reason that "Since CASCADE handles 100% of the problems in its domain which includes 60% of the problems in CANASTA's domain, it has a effectiveness of 60%. Since CANASTA has only a 40% effectiveness, CASCADE outperforms CANASTA." Nor should the development time or cost ratios be merely multiplied by 0.6 to estimate the cost of using our methodology to produce a "full" system. It may well be true that the knowledge needed to decide whether or not a problem falls into CASCADE's domain is, in fact, the hardest knowledge to acquire — thus, our methodology may not scale in this sense. At the same time, our ability to use an existing database allowed us to recognize that 60% of the cases are related and can be easily transformed into a single highly reliable expert system. It is possible that the remaining 40% of the cases can be easily encoded into one or more additional expert systems, and a new problem could be submitted to all the systems concurrently to locate relevant cases. We have not explored this area.

Notice that we have included maintenance time for CANASTA but we have no comparable number for CASCADE. This is not accidental. Recall our definition of maintenance: it is the time taken, above and beyond knowledge

[2]With our 20/20 hindsight we now realize that we should have measured this time separately.

acquisition and encoding, to add new knowledge to a system. In our case-based system there is no such additional time. In a rule-based system, the addition of rules take place in what amounts to an imperative programming language where the sequencing of rule firing is of primary importance. Largely because of the imperative nature of the rules, addition of a rule often requires modification to other rules in order to maintain an acceptable ordering to rule execution. It is this cost that we call "maintenance," and it can be quite expensive. In CANASTA, maintenance already accounts for 17.5% of the cost after only 6 months of use — and this maintenance cost rises with the number of rules, and hence with time.

We now return to the major concern of this paper, knowledge acquisition. In the development of the rule-based VMS diagnosis system, 720 engineer-days of costs went into knowledge acquisition (120 pdays from the knowledge engineer and 120 pdays from the domain expert). In developing our system for the same domain, the comparable number is 120 engineer-days (20 for the knowledge engineer, 15 for the domain specialist, and 5 for the expert). The reduction in cost (a factor of six) is tremendous, as is the reduction in time (a factor of 4). The remainder of this paper explains how this reduction is accomplished: as we have already said, it arises from the use of both an existing database of analyzed cases (to supply data and aid in knowledge acquisition) and a validation model (to encode knowledge).

5 The validation model

We describe elsewhere[13] our two-stage methodology for case-based reasoning. It involves an initial SFB retrieval followed by a justification phase based on deeper knowledge. It is acquiring the domain-specific knowledge for this latter phase that concerns us in this paper. The knowledge we require describes tests that can be performed on new problems for comparison with results of similar tests that were performed on cases in the existing case base. We encode this knowledge in a data structure we call a validation model.

We now describe the validation model in a bottom-up manner. At the very lowest level it consists of individual actions that can be performed on new problems to produce data values. These actions correspond to the simplest kinds of tests that can be performed; for example: reading a value in a known location in memory, measuring the amount of fuel in the gas tank, or taking a patient's temperature.

At the next level, these actions can be grouped together into **probes** to yield less detailed knowledge about the new problem, but knowledge which is more descriptive and hence likely to be more indicative of the kind of difficulty that has been encountered; each action can participate in more than one probe. In the car example we might have one probe to determine if the "fuel injection is working" and another to detect whether the "fuel is low". Both of these probes will require using the action "measure the amount of fuel in the gas tank."

But probes do more than simply group together actions. They summarize the information provided by the constituent actions into a canonical form that can be compared with the summaries stored in the case base. The precise form of the summary is dependent on the purpose of the probe, so two probes that are composed of the same actions may yield very different canonicalizations. For example, in medical diagnosis we might have a probe designed to answer the question "does the patient have a fever?" and another for "is the patient hypothermic?" Both of these depend on the action of taking the patient's temperature, but they canonicalize this knowledge (into a boolean value) in very different ways — the first will be true for temperatures above 99.5°F, the second for values below 97°F.

Probes also produce "interesting information" that was learned in the process of computing the canonical value, and this information is made available to other probes as an **information block**. For example, the exact temperature or the exact amount of fuel in the gas tank might be made available in addition to the abstracted information ("fever," or "fuel low"). These information blocks are an explicit part of the validation model, and act as a cache for intermediate information. When a new problem is examined, all of the information blocks in the model are cleared, and they are filled in as probes execute. Each information block specifies a probe which can be executed to fill in its contents.

To accomplish all of this, probes actually consist of two components. The first is the ordered set of actions to be performed. The second is what we call the **interpretation knowledge**. This takes as input the values computed by the actions as well as the information blocks from earlier probes. It produces two outputs: the canonical value of the probe, and an information block for use by subsequent probes. The fact that one probe requires the information block from another probes implies an ordering to the execution of these probes. When a probe runs, it checks all of its input information blocks to see if they are filled with data from the current problem. If not, it runs the appropriate

probe to fill the block. For example, we may wish to execute a probe to determine whether a patient has an infection. This requires both determining that the patient has a fever and that the white blood count is high. The former is a probe, and the information block filled in by the fever probe will convey the answer. The latter is composed of a sequence of actions ("take a blood sample," followed by "count the white cells").

So far, we have encoded a precedence ordering on the probes based on the information blocks coupling the output of one probe with the input of others. "Probe-space," however, is more complicated than this. It has a roughly hierarchical structure tbat is related to the conceptual structure of the system the probes are designed to test. That is, some probes correspond to complex concepts (like "infection") and others to simpler concepts (like "fever"). This conceptual structure is unrelated to the precedence ordering, and is not directly useful in justifying new problems. It facilitates both the knowledge acquisition sessions, by providing a structure to the discussions between the expert and the knowledge engineer, and the subsequent maintenance of the system, by allowing the knowledge engineer to quickly locate relevant parts of the validation model. The validation model actually consists of this conceptual structure, each of whose leaves is a set of probes.

Finally, we turn for a moment away from the validation model itself and examine the case base. Each case consists of three components: the surface features used in the earlier retrieval phase, the preferred solution for this case, and what we call the validation procedure which can be used to ensure that the solution for this case is applicable to a new problem. This validation procedure consists of a set of probes to be executed and the canonical values of those probes (each pair of a probe with its canonical value is called a validation step). In a world of perfect knowledge, these probes would form a complete causal chain. In fact, they would provide assurance based on first principles that *if* the probe values of a new problem result in these canonical values, *then* this case is relevant to the problem under consideration. The world, however, is not perfect[3], and we can make no such claim about the actual set of probes. The set of probes (and their canonical values) is chosen by the knowledge engineer from comments found in the original database from which the case base is derived.

[3]This observation, for which we claim no credit, is one reason we believe that causal models are poor candidates for justification in case-based reasoning.

5.1 Creating a validation model

With this background, our knowledge acquisition problem is clear: the knowledge engineer starts with a database of textually described cases, each of which includes a list of surface features, a set of tests that were performed to determine a solution, and the solution itself. The job is to produce a case base with the same surface features, and a set of validation steps (probes and canonical values of those probes), and a solution. That is, the knowledge engineer must discover the set of probes (actions and interpretation knowledge) implicit in the database, and then encode each case using this information. Stated another way, the knowledge engineer must first develop a validation model and then encode the cases using that model. The following steps encapsulate a methodology which we have found to be very effective for managing the knowledge acquisition process when trying to develop a new validation model:

1. **Read** each case and build a list of the tests that are explicitly mentioned, along with values derived from these tests. In the process of reading the database and preparing this list, a sense of the underlying (but unstated) relationships between tests is developed. This sense will help determine the probes that are needed for the validation model, the contents of their information blocks, the ordering of the probes, and a reasonable set of canonical values.
2. **Examine** the list of tests, attempting to form related sets of probes. This organization eventually becomes the conceptual structure of the validation model.
3. **Refine** the conceptual structure. This is done initially by consulting standard reference works about the domain, subsequently by interactions with domain specialists, and finally by interactions with domain experts.
4. **Iterate** the above two steps. The final validation model consists primarily of entries corresponding directly to information that appears in the original database. As this iteration proceeds, some items will move from probes to conceptual structure or vice versa; some will move from probes to actions and vice versa; and the precedence ordering of probes will become clear.
5. **Integrate** the probes into the conceptual structure, forming the final validation model. This model should be sufficiently clear that the domain expert can examine it and provide verification that it appears correct.

5.2 Extended Example

Let's take a very simple example of a case database and trace through our knowledge acquisition methodology. We'll work in the domain of automotive diagnosis[4]. Assume a database consisting of the following three cases:

		CASE 1			
make:	MAZDA	**model:**	626	**model year:**	1988
engine type:	2.0L EFI	**miles:**	10,000		
problem:	engine does not start.				
validation:	The fuel injector was clogged.				
solution:	cleaned the fuel injector.				
		CASE 2			
make:	MAZDA	**model:**	626	**model year:**	1984
engine type:	2.0L	**miles:**	60,000		
problem:	engine does not start.				
validation:	The car had a faulty gas pump.				
solution:	Replaced the gas pump.				
		CASE 3			
make:	MAZDA	**model:**	626	**model year:**	1987
engine type:	1.8L	**miles:**	20,000		
problem:	engine does not start.				
validation:	A leak existed in the gas line.				
solution:	Fixed the leak.				

Before we begin knowledge acquisition, we partition the information in each case into surface features and deeper knowledge. In this example, the fields `make`, `model`, `model year`, `engine type`, `miles`, and `problem` constitute (we assume) surface features. Only `validation` is deeper knowledge that we must encode as a validation model. We begin by reading each case, and develop the following list (to the left of the arrow ($\rightsquigarrow$) we list the text from the case, to the right we list our assumption about an appropriate test to perform and canonical value for that test):

1. The fuel injector was clogged $\rightsquigarrow$ `condition-of-fuel-injector`; `clogged`, or `normal`

[4]For a more elaborate example from the same domain, listen to "Car Talk" on National Public Radio.

2. The car had a faulty gas pump $\rightsquigarrow$ `condition-of-gas-pump; faulty, normal`

3. A leak existed in the gas line $\rightsquigarrow$ `condition-of-gas-line; leak, normal`

Here, we've simply assumed that we'll figure out how to probe for each of the conditions mentioned in the case base, and taken a guess (based on our command of English) about the results of such a probe. We don't yet know how to actually perform the probes, nor are we sure that these are the real values that one would discover.

The next step in our methodology is to **examine** the list and attempt to build a conceptual structure. In this example, unfortunately, there really isn't any obvious structure. We do notice, however, that a car has at least three components: a fuel injector, a gas pump, and a gas line. Furthermore, we know (or at least conjecture) that we will have one probe for each of these components. We still have no clue as to the actions needed to build our probes. We build a conceptual structure that has these three components, and has one probe for each of them.

Step 3 is to **refine** this structure. We go to the local book store and purchase a copy of Chilton's Guide for the make of our car. Our major interests at this point are figuring out how to implement each probe from simpler actions, and verifying the correctness of our conceptual structure . Our reading informs us that to determine the condition of the fuel injector the following actions need to be performed (and in this order!):

1. Open the hood of the car.
2. Unscrew the cover of the fuel injector.
3. Remove the nozzle that connects the fuel injector's reservoir with the combustion chamber.
4. Pour water through the nozzle and observe its flow.

The first three of these steps are straightforward, but the fourth one is rather hard to interpret. We need to know just how much flow constitutes (our postulated) `clogged` or `normal` values. For this, we **iterate** by locating a (hopefully friendly) domain expert. After a short discussion, we discover that the expert really has three, not two, values, and add `restricted` as an interesting possibility. We also find out what are reasonable rates of flow that correspond to each of these values.

In the course of this same conversation (or maybe during a later iteration), we discover that there is additional information gained by unscrewing the cover of the fuel injector: we can see if there is any fuel in the reservoir that feeds into the nozzle.

In a similar manner, we proceed to build up a collection of actions that constitute our set of three probes (as it happens, we picked the correct set of probes initially — this doesn't happen so often in practice). We also learn about the precedence constraints between the probes and their interconnection by information blocks.

After a sufficient number of iterations, we will converge on a conceptual structure that is sufficiently detailed for our needs and which a domain expert can verify as matching his understanding of the overall system. We then **integrate** this structure and our collection of probes, to form the full validation model. It is probably wise to have the domain expert verify this final model as well as the conceptual structure on which it is based.

We can continue just a small bit further by considering a knowledge maintenance problem. At a later date, we might become interested in the combustion chamber mentioned in our third action. All we need to do is to modify our conceptual structure to include the chamber — a simple, localized change. In doing this, we will probably realize that the amount of fuel in the injector's reservoir is important for probing the combustion chamber. This is easily encoded: we create an information block that allows the existing probe to propagate the amount of fuel to new probes associated with the combustion chamber.

5.3 How it helps

We have claimed that our methodology reduces the time to produce an expert system by simplifying the knowledge acquisition process. We have shown figures to support this claim, and we have explained the methodology itself. This section discusses a slightly more abstract issue: which aspects of our methodology simplify the problem?

First, our methodology doesn't attempt to automate the knowledge acquisition process (Section 7 discusses some ideas for automating part of the process). Instead, it aims at making the process more efficient, especially with regard to the amount of time required of a domain expert. We do this by significantly altering the role of the expert in the knowledge acquisition pro-

cess. Essentially, we gain leverage off of the existing database. By having a knowledge engineer peruse this database and other knowledge sources, we can have the engineer produce a proposal for a conceptual structure of the expert's domain, based on the expert's own vocabulary, as recorded in the cases. Just as it is far easier to critique and edit a document than to create one, it is similarly easier for the expert to examine the proposed structure than to create one from scratch. Thus, we simplify the task that we ask the expert to perform exclusively for making the expert system. (We assume that the expert's normal duties include maintaining the database, and we exclude this time from our consideration.) It would be reasonable to expect *any* case-based methodology to simplify the knowledge acquisition problem in this way (in fact, this is an early claim in support of these methods). In practice, however, other case-based reasoning systems continue to rely on causal models whose acquisition remains very costly.

Secondly, we focus the attention of the knowledge engineer by providing specific knowledge structures (probes, actions, information blocks, conceptual structure). The case database also provides the initial seed for all of these structures. This point is important: in most design methodologies, the initial steps are the hardest to take, require the most thought, and often have the greatest penalty attached to small errors. Thus, any reasonable person (including a knowledge engineer) is very hesitant about taking them. Our methodology starts very simply: read the existing database, and write down the tests you see being used. The method of iterative improvement gradually guides the engineer's steps toward a working system structure, rather than requiring a carefully designed initial structure. In addition, our methodology has a self-contained measure that indicates when the process should stop: when the validation model contains all the probes needed to encode the existing cases, and a domain expert agrees with the overall precedence constraints.

Finally, we will raise an issue not directly related to knowledge acquisition, but perhaps the most vital in calculating the cost of any expert system: the problem of maintenance of the knowledge. The validation model is inherently structured, since it provides probes for grouping actions, a conceptual hierarchy to organize probes, and explicitly represents dataflow constraints through the use of information blocks. But modularizing constructs alone are only of theoretical interest; our methodology guides the knowledge engineer toward a practical use of the constructs. Since the validation model is repeatedly examined by domain specialists and experts, it gradually becomes structured in

31-14

a manner they find acceptable for explaining their domain. On rare occasions, a major breakthrough may alter the experts' own perception of the domain's modularity, but these breakthroughs are extremely rare and will invalidate any representation of domain knowledge (consider the invention of the telegraph with respect to international banking). Much more often, however, the knowledge maintenance problem is really one of making a small addition or modification to the knowledge base. If the base is organized in a way compatible with the experts' view of the domain, then the change will almost certainly be relatively localized. It may require an expert to state where the change should occur or the precise nature of the change, but the fact that the place exists (and hence can be located) is an immense aid in the maintenance process.

6 Results

We have implemented two different expert systems using our methodology. CASCADE, for diagnosis of VMS crashes related to device drivers, was mentioned earlier and contrasted to the rule-based system CANASTA for a related domain. The second is for diagnosis of problems with the WPS–PLUS word processor. This section contains details of the implementation times, knowledge acquisition times, and performance of each system. In order to fully present each system, we provide information about both the initial retrieval based on surface features and the subsequent justification using a validation model.

6.1 Device Drivers

Information about surface features was obtained primarily from DEC internal publications and was complemented by a domain specialist from the VMS support team during three knowledge acquisition sessions. It took a total of 11 pdays to acquire the domain specific knowledge about surface features. This knowledge was implemented in five days.

It took an additional four days of reading the database and interacting with domain specialists to develop a validation model. In addition, four knowledge acquisition sessions with a domain expert, lasting five days, were needed to refine and improve the validation model. Encoding the actual validation model

took 80 days. The total number of days spent on knowledge acquisition (KA) and development is shown below:

Activity	Person Days
KA (expert)	5
KA (specialist)	15
Development	85

The system was evaluated using a case base of 200 cases that were obtained from notes written by specialists. The SFB retrieval phase of the system was evaluated by presenting each of the 200 cases to the retriever (as new problems). UNIMEM[10], the algorithm that we have used for SFB retrieval provides a mechanism, known as **retrieval weights**, for tuning its retrieval capabilities. After some experimentation, we discovered that the use of larger retrieval weights (i.e. more stringent matching criteria) caused the retriever to miss many relevant cases and, on many occasions, to fail to retrieve any cases at all. With less stringent criteria this problem was rectified. However, many of the retrieved cases were not relevant to the problem. With the best weightings we found, we were able to retrieve on average 22 cases per retrieval (11%). The validation phase, however, was able to reduce this number of cases to an average of 4.5 cases out of 200 (2.25%).

In addition, we presented three new cases to the system. Based on SFB retrieval alone, we found 20, 25, and 16 cases (10, 12.5, and 8% selectivity). The validation phase reduced this to 3, 5 and 3 cases, respectively (1.5, 2.5, and 1.5% selectivity). Our experts confirm that these validated cases are the only ones relevant to the problems presented.

6.2 A Word Processing System

The second system performs diagnosis of customer problems with the word processing component of an office automation product. Information about surface features was acquired from a domain specialist and from internal publications within five days. It then took an additional five days to encode this information and use UNIMEM to develop a generalization hierarchy for SFB retrieval. We use a database with 340 cases.

The validation model was obtained from the cases in the database, an internal publication, and five days of knowledge acquisition with a domain expert. It took 40 days to implement the validation model.

The time we spent on knowledge acquisition and development is shown below:

Activity	Person Days
KA (expert)	5
KA (specialist)	5
Development	40

We have evaluated the system using the same method as in the first prototype. Our experiments showed that on an average 26 cases are retrieved during SFB retrieval, a 7.6% selectivity. The validation phase reduced this to an average of two cases, or 0.58% selectivity.

7 Automating the Acquisition Process

Our current methodology, while effective, is entirely manual. Because of the complexity of validation models, and because the input to our process is expressed in highly technical language, we do not expect to be able to automate the entire process. At this time, we have not yet explored any automation methods, although we see four very promising avenues to be pursued.

Corpus-based NLP.

The first step of our methodology is to read the entire database. Our largest database contains 340 cases described in about 500 pages of clean text; the initial reading of this database required five days. As the database becomes larger, this task becomes harder. By extracting the textual descriptions of how a case is validated and comparing this text with a standard English corpus (perhaps augmented by standard computer science jargon), we should be able to automatically produce a list of words that are good candidates for consideration as names of concepts, probes, or actions. If the NLP tool can refer each word back to a syntactic category and/or to the source text in which it occurs (along with frequency counts), this would be even more helpful. It would be unreasonable to replace the scan of the database entirely by automated tools, however, since part of the purpose of that scan is to help the knowledge engineer begin to understand the domain before the initial meeting with a domain expert.

Deducing precedence constraints.

If we make the (reasonable but not necessarily correct) assumption that the text of a case indicates tests to be performed in the order in which those tests are actually performed, we can combine these ordered sets to produce a proposed partial ordering of all the probes mentioned in the database. For each proposed ordering constraint the knowledge engineer would then produce an information block and ask whether data actually flows between the probes. (It is inherent in our model that the precedence constraints between probes are derived from dataflow constraints. There may be other types of constraints needed in other domains: costs of tests or destructive tests, for example.)

Deducing conceptual structure from probe sets.

In a process reminiscent of UNIMEM's concept formation, we can take the set of probes associated with each case and look for commonly occurring subsets. Each such subset can then become a candidate for forming a higher level concept, and the new concept replaces the earlier subset in appropriate cases.

Deducing conceptual structure from action sets.

A slightly more complicated process that extends the previous idea attempts to form new probes from commonly occurring subsets of *actions* within existing probes. This is more complicated because a probe uses its inferential knowledge to combine the results of the actions to form a single canonical result. In order to successfully combine common actions into a single probe, we would have to be sure not only that the actions all occur together in existing probes, but that the inferential knowledge combines the results in the same way in those probes. Thus, the same subset of actions might be combined into multiple *distinct* probes, based on how the action results combine to form a canonical result.

8 Conclusion

We have presented a methodology that dramatically reduces both the time and the cost of producing an expert system. In addition, the maintenance time associated with a rule-based system (the time required above and beyond that needed for acquiring and encoding new knowledge) is reduced to a negligible level. In one comparison, we reduced the time required to produce an expert

system by a factor of four and the cost bya factor of six.

These improvements arise from three aspects of our methodology:

1. The methodology changes the roles of the knowledge engineer and the expert. The knowledge engineer begins by reading an existing database of solved problems and summarizing the information in the form of a conceptual model. The expert is asked to evaluate this conceptual model and provide feedback to the knowledge engineer. Thus, the engineer plays a much more active role in the process than in traditional rule-based system development, and the expert's time is more effectively utilized.

2. The methodology specifies the type of knowledge which must be assembled: information about tests that can be performed (actions and probes), and the relationships between them (conceptual model and dataflow constraints). This not only focusses the energy of the knowledge engineer, but it provides us with hope that part of the acquisition process can be automated. Since most of the work is centered around examining an existing database, we can use automated tools to examine the database for specialized terms that are good candidates for inclusion in the conceptual framework of the validation model.

3. The validation model consists of a standard set of data structures, so the knowledge engineer does not need to design data structures. This eliminates one major barrier in knowledge engineering: that of representing the acquired knowledge. Another major barrier, more psychological than technical, is also eased. Since our methodology is one of iterative refinement, it isn't necessary to aim immediately for "the correct structure" of the concept space. Any reasonable initial structure will serve well; this reduces the activation energy needed to get the implementation process under way.

We see three major directions for our future work. The first is to begin automating some of the steps in the methodology, along the lines discussed in Section 7. The second is to look at additional domains, farther afield from computer science, and verify that our methodology works well when the knowledge engineer is less familiar with even the high-level structure of the application domain. We are particularly interested in domains, like law or psychoanalysis, where there is reason to believe that a causal model cannot be constructed; we conjecture that case-based reasoning can work even in these domains. We are also interested in exploring our methodology when applied to problems other

than diagnosis: we have built scenarios involving sales and management that appear to be amenable to case-based reasoning. Finally, we are interested in examining significantly larger databases — perhaps a database with 2,000 to 3,000 entries. As the number of cases grows, our methodology must certainly receive automated help (no engineer is likely to have time to read all of the cases). But, more importantly, it will almost certainly be necessary to deal with experts from different areas. Will it be most efficient to subdivide the cases to correspond to the different areas (as we've done by considering only device-driver failures in VMS), or will such a subdivision become unwieldly? How hard will it be to develop a single conceptual model that experts from different disciplines can understand (for example, imagine the validation model for an airplane where experts range from mechanical, to material, to electrical, to aerodynamic).

References

[1] John R. Anderson. *The Architecture of Cognition.* Harvard University Press, 1983.

[2] Ray Bareiss, Karl Branting, and Bruce Porter. The role of explanation in exemplar-based classification and learning. In *Proceedings of Case-Based Reasoning Workshop*, 1988.

[3] J.S. Bennett. Roget: A knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Automated Reasoning*, 1:49–74, 1985.

[4] Randall Davis. Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, 12:121–157, 1979.

[5] Ashok Goel and B. Chandrasekaran. Use of device models in adaptation of design cases. In *DARPA Workshop on Case-Based Reasoning*, pages 100–109, 1989.

[6] Kristian Hammond. *Case-Based Planning: An Integrated Theory of Planning, Learning, and Memory.* PhD thesis, Yale University, 1986.

[7] Subbarao Kambhampati. Representational requirements for plan reuse. In *DARPA Workshop on Case-Based Reasoning*, pages 20–23, 1989.

[8] Phyllis Koton. *Using experience in learning and problem solving.* PhD thesis, Massachussetts Institute of Technology, 1988.

[9] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[10] Michael Lebowitz. Experiments with incremental concept formation: Unimem. *Machine Learning*, 2:103–138, 1987.

[11] T. Michell, R. Keller, and S. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1, 1986.

[12] Steven Minton. Qualitative results concerning the utility of explanation–based learning. In *Proceedings AAAI-88*, pages 564–569, 1988.

[13] Evangelos Simoudis and James Miller. Validated retrieval in case–based reasoning. In *Proceedings AAAI-90*, pages 310–313, August 1990.

[14] Katia P. Sycara and D. Navinchandra. Index transformation and generation for case retrieval. In *DARPA Workshop on Case-Based Reasoning*, pages 324–328, 1989.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Ontological Issues in Knowledge Base Design: Some Problems and Suggestions

Doug Skuce[1]

Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada

Ira Monarch

Philosophy Department
Laboratory for Computational Linguistics
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

As more and more researchers and practitioners engage in constructing knowledge bases, questions concerning their reuse, sharing, and meaning are becoming increasingly important. We believe that the underlying ontological assumptions, insofar as they are uncritically applied or not made explicit can impose significant limitations on the usefulness of the resulting knowledge base. Not only must the ontology be made explicit and understandable, there must be well defined ways of reaching agreement on its nature and content. Questions like what kinds of scientific research can contribute to ontology construction, how ontological categories should be characterized, and what the meta-terminology should be needs to be discussed, partly with a view to standards being established. Methods for establishing agreement about what the top level categories and relations should also be proposed and tested. We will identify a number of such key issues and will offer some recommendations regarding *methodology for ontological design* in particular.

---

[1] email: dsksl@uottawa.bitnet

1 INTRODUCTION

As designing and constructing knowledge bases becomes a more and more common activity, questions concerning the extent to which a knowledge base can be reused, shared, extended consistently, and indeed *properly understood* are becoming increasingly important. We believe that the underlying ontological assumptions, insofar as they are uncritically applied or, worse, not made explicit, can impose significant limitations on the usefulness of a knowledge base. Not only must the ontological assumptions be made explicit, there must be explicitly defined ways of reaching agreement on its nature, content, and how it was created. We are concerned with questions such as:

* what kinds of scientific research are useful to guide ontology construction?

* how should ontological categories be characterized?

* what should be the meta-terminology and naming conventions, with a view to standards being established?

* what is the linguistic and logical interpretation of the contents of typical "frames" found in knowledge bases?

* what methods can be found for establishing agreement about what should be the top level categories, relations, and the methodology for determining them?

We have been studying some of these basic questions in ontological design, focussing on questions of methodology. We believe that many important ontological issues are not being given adequate consideration, and hence some knowledge bases may become white elephants. Failing to recognize these issues and hence to discover solutions may lead to an ever-increasing number of knowledge bases being produced, each with its own idiosyncratic assumptions, terminology, methodology, and underlying ontology. For example, we have never seen what we would consider a properly documented ontology. We feel the current situation is reminiscent of the early days of programming, in which there was no discipline or

methodology: people wrote programs by the seats of their pants, and hence there was much reinvention of programs as people found it easier to recreate rather than to understand opaque code. We therefore seek to banish opaque ontologies[2].

2 THREE MODELS FOR KNOWLEDGE BASES

We have found it useful to think in terms of three "models", which we term the "dictionary model", the "encyclopedia model", and the "thesaurus model". By this, we mean that a knowledge base is in some ways analogous to:

* a *dictionary*, in which ideally all words of a language are isolated, interrelated, and crucial information about them is presented compactly and in a fixed format. A dictionary "defines" a term in one of two ways: either entirely in terms of other terms used without restriction, or it may use a restricted vocabulary, such as the LDOCE (Longman78), in which a set of some 2000 terms or concepts is selected as a defining vocabulary (however they themselves are used to define each other circularly). Some dictionaries include "world" or "encyclopedic" information as well, blurring the distinction between a dictionary and an encyclopedia.

* an *encyclopedia*, which organizes knowledge by topic, and gives great detail. It makes only secondary attempts to define meanings, and is far from comprehensive.

* a *thesaurus*, which emphasizes conceptual taxonomies and semantic proximity (synonymy). Terms are not defined but classified in appropriate branches of the taxonomy; distance in the taxonomy corresponds roughly to semantic distance. There is a notion of definition, but entirely in terms of circular synonymy. The organizational principles of the conceptual taxonomies of Roget77 have much in common with what we believe ought to be found in taxonomies for knowledge bases.

---

[2] There has been virtually no emphasis on making ontological assumptions explicit in the literature. Regoczei&Plantinga87 have discussed the importance of ontological matters. Others have recommended making tacit knowledge explicit, especially with respect to rule based systems (e.g. Clancey83). Others have recommend clarifying the semantics of knowledge representations (Brachman85, Woods85). We go considerably further in recommending the explicit specification of all ontological assumptions.

Knowledge bases have characteristics of all three. Most designers of knowledge bases have shown little interest in these distinctions, terminology, or problems associated with these three classical forms of structured knowledge, but part of our approach (due to our background in language-intensive applications of knowledge bases) is to draw inspiration from some of these similarities.

In this paper, we will attempt to elucidate a number of the key issues in ontology design, using the three models cited above as a point of reference. We then will make some recommendations about how to approach questions of *methodology* in particular. The paper is organized as follows: In Section 3 we introduce some of our meta-terminology; in Section 4 we discuss some major issues other than those concerning methodology; Section 5 describes our linguistically-oriented methodology, while Section 6 points out some additional methodology issues that apply general. Section 7 offers concluding remarks.

3 SOME BASIC CONCEPTS AND TERMINOLOGY

First we clarify our use of some key terminology. We use the term *knowledge* to mean domain specific knowledge that experts in the given domain agree on. By *ontology* we mean the very general, usually tacit, views about basic semantic categories (e.g. entities, properties, situations) and their properties that people share. Because ontology is usually tacit, a key problem (on which we will focus) is how to establish ways of uncovering these assumptions and rendering them explicit, for only thus can we hope to reach agreement among and between knowledge base designers and users. Although an initial impulse of many upon first encountering the idea of ontology is to think that everyone has a different ontology or that there are as many ontologies as points of view, we believe that people's ontological views will have much in common once they are made explicit through appropriate analysis and criticism.

Knowledge bases are computer repositories containing domain specific knowledge, employing anything from vague, unconscious ontologies to very precise ones. By an *ontology base* we mean a well defined set of very general top-level *semantic categories* and associated *relations* and *assumptions* that makes the ontology as

explicit as possible. An ontology base, which should precede knowledge acquisition, may be open to modification as knowledge acquisition proceeds, though too much modification would reduce its effectiveness as the key medium for explicating the ontological assumptions behind the knowledge representation, the knowledge itself, and the acquisition process. Thus one adds domain-specific knowledge "below" an ontology base, and sharing and reuse is most enhanced when the ontology base becomes stable.

We suggest that it may be possible to create a *general* ontology base, general enough to be useful in virtually any subject area, i.e. which does not include any knowledge specific to some area. From such a base, *restricted ontologies*, e.g. for medicine or electrical engineering can be constructed. By this we mean an ontology base extended to contain the general concepts for the domain, i.e. that are subcategories of the higher, totally general ontological categories. This is the main way that a general ontology base guides knowledge base construction. It helps to ensure that the knowledge base can be used for various purposes in the same domain, or when coupled to others in different domains that share the same global ontology. Since such ontology bases are as yet not available for reuse, knowledge base developers, including ourselves, are constantly having to "start from scratch", designing and redesigning ontology bases without any agreed upon standards or methodology. These ontology bases usually are constantly being modified as new knowledge is added, causing serious consistency and reworking problems. Thus, one is not able to build effectively on other peoples' work.

Another term that concerns us is "primitive". In general, the term *primitive* usually refers to a semantic category, relation, or term that is used to define or describe others. It may or may not be defined circularly using other primitives. Sometimes, as is the case with the defining vocabulary of the LDOCE, a restricted set of terms or concepts is selected constituting a controlled vocabulary. (LDOCE nevertheless defines each primitive in terms of the others.) While the top level categories that form an ontology may be called primitives by some, we prefer to call them "the top level categories" and reserve the term "primitive" mainly for relations, such as those termed "epistomological" by Brachman.

Two other closely related key terms are *category* and *concept*. We suggest that the former apply to knowledge units in the ontology

base, while the latter apply to all other units (lower in the knowledge base). Thus semantic categories are just high level notions which necessarily are defined in terms of other peer level categories. What *characterizes*[3] a category then is its relationship to other categories, whereas concepts may rely both on the higher concept level, and each other.

A *knowledge unit* is a collection of elementary assertions about one subject. It may correspond in "size" to a paragraph or two in natural language, and often is represented as a frame structure in a computer. A clear assertion regarding the question of definition or characterization we term the *definitional status* of a knowledge unit.

4 MAJOR NONMETHODOLOGICAL ISSUES IN ONTOLOGY

We discuss next some of the major issues other than methodology surrounding the design and use of ontologies. This list is far from exhaustive; space and time restrictions led us to focus on some of the most crucial. In Section 5, we focus on methodological issues.

The question of what should be the most general (top level) semantic categories[4] goes back at least to Aristotle66, who proposed some that today remain very plausable. In the nineteenth century, Roget proposed a very exhaustive set of categories for his thesaurus (Roget77), though their meaning was only implicitly conveyed by listing similar terms in the same category. Among ai researchers, we know of only two proposals for general ontologies that have appeared in books, those of Dahlgren88 and Lenat90., and one that has appeared as a techical report (BatemanEtAl90) In Dahlgren, details of many important categories have not been described. Lenat does attempt to describe informally most top level categories[5]

---

[3] We prefer using the term "characterize" rather than "definition" when talking about the meaning of a category or concept in a knowledge base.

[4] Most concepts are fuzzy, yet to date, the vast majority of knowledge acquisition systems use knowledge representations that are strictly symbolic and non-fuzzy. The recent movement toward connectionist representations is an extreme reaction to this serious limitation. Since it may be many years before connectionism or other approaches to fuzziness are sufficiently developed, some accomodation of fuzziness may be necessary in existing symbolic models. However, our goal is to design the top level categories as crisply as possible.

[5] We plan a comprehensive study of these and the methodology of their exposition.

however, neither is explicit about methodological matters. The BatemanEtAl work derives from the systemic grammar theory of M. A. K . Halliday85, and is the only one to be based on sound linguisitic knowledge. While each category is described, it is difficult to follow without reading Halliday85. Lack of adequate documentation and lack of attention to methodological issues are key issues that need to be resolved. We would like to see each category explained as clearly as possible (admittedly not an easy task), using carefully chosen and consistent terminology, together with the rationale for choosing the category. Ideally, the complete text of each knowledge unit, suitably explained at least in English, should be available.

Here is a list of some of the terms that are commonly used to name the most important categories, as we see it (object level terminology):

concept, object, thing, entity, situation, relation, property, attribute, state, event, process, activity, action, part, whole

Some typical meta-level terminology includes:

type, class, intension, extension, instance, token, individual, referent, predicate, proposition

We look forward to the day when this and similar terminology will be used much more clearly and consistently than at present. We would hope that soon, we will be able to present our proposals for some definitions for such terms.

We do not have space to discuss some other ontologies that have been proposed. Some other representative examples of general ontologies may be found in Wierzbicka85, NirenbergEtAl88, Barwise and Perry83 None of these except Wierzbicka include any attempts at explaining the intended meaning of the categories or the methodology. Examples of restricted ontologies, which include more explanation of categories, are HobbsEtAl88 and Hayes88.

Another key question is whether the notion of *semantic primitives* is useful at all, and if so, what this term should mean. The term is used in various ways; some use it to mean the top level categories, others, to mean primitive binary relations, often associated with semantic network representations, the sense we use here (Woods85, Brachman85, Sowa84). From a linguistic point of view, only a few sets of primitives that might be termed "semantic" have been

proposed, which we cannot review here (Wilks77, Schank&Riesbeck81, Melcuk89). A methodology should make clear whether or not it considers that parts of knowledge units constitute some type of definition or characterization, and whether or not some notion of semantic primitives (either restricted to relations or in the more general sense) is being used. Some researchers believe that it is either hopeless to seek such primitives at this point, or that it may not even be desirable, i.e. that knowledge bases should look more like thesauri than LDOCE-style dictionaries. But only by attempting to find such primitives will we eventually learn if this approach is fruitful.

The knowledge representation chosen also has a major impact on ontological issues. In most cases, some fixed (or often evolving) knowledge representation is prescribed (e.g. CYC and KBMT89 (Goodman89)); in others it is left open-ended. The degree to which the limitations of the knowledge representations affect ontological assumptions is, we suspect, often underestimated. For example, users of frame-based representations may make the assumption, consciously or not, that all relations (represented as slots) are binary and have a "directionality", i.e. a "from" and "to" argument that conceptually may not make sense in that both arguments should be equal partners. Another undesirable commitment may be that all attributes must have a value (e.g. the value of "shape" = "square"; the value of "sex" = "male"). A third common cause of confusion comes from using the same "frame" syntax to represent both "class" and "instance" knowledge. The meaning of the two is quite different, but the frame syntax may look identical. We believe that such decisions should only be made when it is realized a) that they may be an artifact of the knowledge representation, b) what are the alternatives, c) what are the consequences.

A final important point is that an ontology base should be able to assist in a semantic classification of closely synonymous terms in not only one but *various* natural languages and, in that sense, should form the basis of an intertranslatable thesaurus. Hence it is important to establish standards and methods for reaching agreement upon a set of ontological categories, all the more so if we concern ourselves (as we should) with translatability.

Our Maxim:

Without a clear understanding of the ontology base, any knowledge base is open to misunderstanding and improper use.

5. TOWARDS A LINGUISTICALLY BASED METHODOLOGY FOR DESIGNING ONTOLOGIES: AN EXAMPLE

5.1 Introduction

We have been trying to simultaneously design a general-purpose ontology base and identify the underlying issues like those we raised above. We have focussed on trying to introduce as much knowledge as possible from the linguistics and cognitive science literature[6] into the process, since we believe it can offer the most assistance in making ontological choices. Thus we have two goals:

a) to produce an ontology base that has a firm basis in existing ontological and linguistic principles, that is general and useful, that is well documented, and that has been validated (see 6.2) by a significant set of knowledgable users or potential users. We view this as a several year project.

b) to report in particular on our approach to the methodological issues.

We are not yet ready to discuss the former, and this paper is a first attempt at the latter. In describing our methodological approach, we will refer to only one representative branch of the ontology, the 'attributes'.

One of the difficult ontological categories is the so-called *attributes*, also often termed "properties", "qualities", or "features". To give the reader a quick idea of where attributes fit in our ontology, we have three main top level categories: entities, situations, and properties.

We have two subcategories of property, the *relations* and the *attributes*. Relations involve two or more things on a roughly equal status, whereas attributes involve primarily only one, which they

---

[6] Shaw and Woodward89 emphasize the importance of drawing upon the cognitive science literature for clarifing the notion of a "conceptual model", which is very similar to our notion of "ontology".

supply information about. Syntactically they originate mainly as adjectives and adverbs. We cannot go into here our detailed analysis of attributes, but we have tentatively identified a half dozen or so subcategories and associated primitives.

We discuss next some of the methodological issues we encountered in our analysis and our response to them.

5.2 Consult the Linguistic and Cognitive Science Literature

We had great difficulty in trying to devise a conceptual structure for attributes, and in finding reports of such attempts by others on which to build. We found virtually nothing useful in the ai literature. Most ontologies in existence in ai have usually been built for very practical purposes. The designers probably did not make any deliberate attempt to incorporate concepts from disciplines like linguistics, psychology, and philosophy, relying instead on what they might find in the ai literature and on their own intuitions about language. This is understandable given that

* people may not feel that these areas are relevant (e.g. there is relatively little mention of the importance of linguistics in the knowledge acquisition literature)
* it is not clear where and how to incorporate such ideas
* the needed concepts are still very vague and controversial (e.g. reading the philosophy literature can be a frustrating exercise for non-philosophers, yet it is there that the most fundamental discussions of ontological problems are to be found)
* most ai workers do not know this literature, which is often hard to find, let alone understand. They may not even know someone who could help them.

In our attempts to decide on semantic categories and in particular to discover semantic structure for attributes, we consulted the linguistic, cognitive science, and philosophy literature. For some major semantic categories, like situations, events, or processes, we have collected many useful articles. However there has been relatively little work on attributes, but we have discovered a few (e.g Warren84). Our point is that the linguistic literature was virtually our only source of ideas for ontological analysis, since, in our view, ontology is intimately associated with language. This reinforced our

view that a language based methodology is a promising approach for establishing an ontological base.

5.3 Standardize Terminology and Naming Conventions

The problem of finding and using consistent and rational terminology, both at the meta and the subject matter levels, is particularly serious in ontological design, since the very essence of such research is to find clear, generally accepted concepts and associated terminology by which to communicate knowledge to others. We recommend trying to maintain and follow glossaries of all technical terminology as carefully as possible.

The *naming problem* is that often there are no criteria for choosing names of knowledge units or their components (e.g. frames and slots). We feel there should be a global dictionary of all names used and that this should be an integral component in any knowledge base system. It could be coupled to a machine-readable dictionary and could enforce certain syntactic conventions. We have established some simple syntactic criteria for the choice of slot and frame names in our systems; we do not allow long phrases that contain "buried knowledge".

5.4 Take a Clear Position on Definitional Status and Primitives

Above, we pointed out the questions of definitional status and semantic primitives, and we return now to these problems. English does not have a good term for the former notion: "the knowledge unit is a _____ for the concept". However the term "definition" is often used by knowledge base personnel, and even lexicographers sometimes refer to the entries found in typical dictionaries as definitions. We believe that this is a specialized meaning of the term, and we prefer the looser term "characterization".

Often a knowledge unit will have many parts (e.g. slots) and some of these might constitute an actual "definition" (if there is one), the rest constituting knowledge that should be termed "encyclopedic", the term usually used by lexicographers. Hence knowledge units (that correspond to a type) should clearly distinguish

* parts that constitute an actual definition or characterization, if there is one
* if appropriate, necessary and typical conditions that instances have
* same for sufficient or typical conditions that imply something is an instance
* any other knowledge that is not about instances

Turning to the semantic primitive question, we may consider two extremes:

1 there is no set of prescribed "primitives", i.e. a knowledge unit may use any combination of already defined concepts and undefined ones in making a new concept characterization;

2 a set of primitives is given, and all characterizations must either use it directly (e.g. like LDOCE), or must use others which can be reduced to the primitives (e.g. like in mathematics).

We suggest that at the moment, neither extreme is desirable. The former leaves the door open to an undisciplined proliferation of terms, both undefined and badly defined (e.g. circularly). The latter will not be possible until much more research has been done on selecting a good set of categories and primitives, assuming one subscribes to this approach. Hence a middle ground should be sought: a partial, initial set of semantic categories and primitives is given, which is under gradual evolution. Knowledge enterers are asked to try to use them as much as possible, i.e. by introducing new concepts only as specializations of top level categories, and to note carefully every place they feel there is none sufficient for some need, so that "holes" in the set can be discovered and covered.

We suspect that the bulk of knowledge base development today follows 1. We believe 2 may be a better alternative, if it becomes possible; it is too soon to tell. It will probably draw much of its inspiration from research in dictionary and thesaurus construction, which is still itself in a "primitive" state. We suggest the two endeavours of finding good primitives for knowledge base construction and good primitives for dictionary construction are two sides of the same problem. Workers in these two fields, who have not been aware of each other in the past, are beginning to cooperate (e.g. Skuce and Meyer89)

5.5 Use Well Defined Corpora

Although this might seem obvious, we believe that some knowledge bases are constructed from a haphazardly selected collection of knowledge sources, and the contribution of each is far from clear. This makes it virtually impossible to predict what knowledge to expect in it, or to verify any knowledge unit. A knowledge base should have associated with it a clear statement of the sources used. Our methodology has focussed on textual sources, which we can cite.

When interested only in a narrow subdomain (e.g. the IBM XT, used in the KBMT89 project), one may define the coverage by collecting all terminology from one or a few well-prepared documents on the specific subject as a guide to the set of concepts that must be captured. Some automated method for accumulating terminology sets should be used (e.g. Szpakowicz90).

However, if one is interested in creating general purpose ontologies (as we currently are), there are two well-defined sources for comprehensive word sets that seem useful:

* a general corpus (e.g. the "Brown" corpus for English (Kucera&Francis82))
* the "controlled vocabulary" used in the LDOCE[7]. For example, in our current analysis of attributes, we are using the adjective set from Johansson75.

5.6 Dealing with Polysemy

Many, if not most, of the words used to name either knowledge units or relations (e.g. slots) in typical knowledge bases are polysemous, without any account of which meaning is intended being made. The creators may not be aware of this, since one meaning is usually fixed in their minds. A similar and surprising example is that much of the defining vocabulary in the LDOCE is polysemous (the reader is expected to disambiguate) (Wilks90). Part of our approach involves identifying such terms and clarifying which meaning is intended, to ensure that they are used consistently. There is no way we know of

[7] Yorick Wilks90 and his associates are currently working to improve this set and to make it more suitable for use in natural language research.

for a machine to identify inconsistent sense uses. To help deal with this serious problem, we would suggest:

* choosing terms with as little polysemy as possible (an online dictionary would help)
* replacing polysemous contractions with full terms that distinguish the meanings[8]
* attaching sense subscripts where unavoidable (these could appear only on request)
* having selectional restrictions and other "type checking" mechanisms built into the system

For the other cases, a system that could pop up the various meanings during review by knowledge validators would permit checking that every use conformed to the intended meaning. This would be laborious, to be sure, but careful knowledge validation unavoidably is.

5.7 Deciding Upon a New Category or Concept

There are two extremes, hut most knowledge entering falls in the middle. Starting from "scratch", in what we call ontological design, someone must create an ontology base, i.e. decide on all the most basic categories and relations (assuming the knowledge representation issue is settled.) This is extremely difficult and crucial: all else depends on it. Only highly experienced knowledge workers should attempt it, nevertheless equally experienced workers will probably come up with quite different ontologies, unless they share the same environment or beliefs (and this does not guarantee that they won't!) We cannot give guaranteed sources in which one is sure to get ideas for these primitive categories and relations, since it is a highly creative endeavour, but here are some we use:

* read the linguistics, philosophy, or psychology literature
* look at other people's knowledge bases/ontologies
* take some domain and try to generalize from it

---

[8] A recent example encountered by one of the authors: the technical term "actor" was causing much confusion, since it turned out to be a contraction for either "actor specification", "actor reference", or "actor incarnation". Another example: a knowledge base in which units for "garbage", "trash", and "rubbish" had heen created without any apparent connections being made between them.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Ontological Issues in Knowledge Base Design: Some Problems and Suggestions

Doug Skuce[1]

Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada

Ira Monarch

Philosophy Department
Laboratory for Computational Linguistics
Carnegie Mellon University
Pittsburgh, PA, USA

Abstract

As more and more researchers and practitioners engage in constructing knowledge bases, questions concerning their reuse, sharing, and meaning are becoming increasingly important. We believe that the underlying ontological assumptions, insofar as they are uncritically applied or not made explicit can impose significant limitations on the usefulness of the resulting knowledge base. Not only must the ontology be made explicit and understandable, there must be well defined ways of reaching agreement on its nature and content. Questions like what kinds of scientific research can contribute to ontology construction, how ontological categories should be characterized, and what the meta-terminology should be needs to be discussed, partly with a view to standards being established. Methods for establishing agreement about what the top level categories and relations should also be proposed and tested. We will identify a number of such key issues and will offer some recommendations regarding *methodology for ontological design* in particular.

---

[1] email: dsksl@uottawa.bitnet

1 INTRODUCTION

As designing and constructing knowledge bases becomes a more and more common activity, questions concerning the extent to which a knowledge base can be reused, shared, extended consistently, and indeed *properly understood* are becoming increasingly important. We believe that the underlying ontological assumptions, insofar as they are uncritically applied or, worse, not made explicit, can impose significant limitations on the usefulness of a knowledge base. Not only must the ontological assumptions be made explicit, there must be explicitly defined ways of reaching agreement on its nature, content, and how it was created. We are concerned with questions such as:

* what kinds of scientific research are useful to guide ontology construction?

* how should ontological categories be characterized?

* what should be the meta-terminology and naming conventions, with a view to standards being established?

* what is the linguistic and logical interpretation of the contents of typical "frames" found in knowledge bases?

* what methods can be found for establishing agreement about what should be the top level categories, relations, and the methodology for determining them?

We have been studying some of these basic questions in ontological design, focussing on questions of methodology. We believe that many important ontological issues are not being given adequate consideration, and hence some knowledge bases may become white elephants. Failing to recognize these issues and hence to discover solutions may lead to an ever-increasing number of knowledge bases being produced, each with its own idiosyncratic assumptions, terminology, methodology, and underlying ontology. For example, we have never seen what we would consider a properly documented ontology. We feel the current situation is reminiscent of the early days of programming, in which there was no discipline or

methodology: people wrote programs by the seats of their pants, and hence there was much reinvention of programs as people found it easier to recreate rather than to understand opaque code. We therefore seek to banish opaque ontologies[2].

2 THREE MODELS FOR KNOWLEDGE BASES

We have found it useful to think in terms of three "models", which we term the "dictionary model", the "encyclopedia model", and the "thesaurus model". By this, we mean that a knowledge base is in some ways analogous to:

* a *dictionary*, in which ideally all words of a language are isolated, interrelated, and crucial information about them is presented compactly and in a fixed format. A dictionary "defines" a term in one of two ways: either entirely in terms of other terms used without restriction, or it may use a restricted vocabulary, such as the LDOCE (Longman78), in which a set of some 2000 terms or concepts is selected as a defining vocabulary (however they themselves are used to define each other circularly). Some dictionaries include "world" or "encyclopedic" information as well, blurring the distinction between a dictionary and an encyclopedia.

* an *encyclopedia*, which organizes knowledge by topic, and gives great detail. It makes only secondary attempts to define meanings, and is far from comprehensive.

* a *thesaurus*, which emphasizes conceptual taxonomies and semantic proximity (synonymy). Terms are not defined but classified in appropriate branches of the taxonomy; distance in the taxonomy corresponds roughly to semantic distance. There is a notion of definition, but entirely in terms of circular synonymy. The organizational principles of the conceptual taxonomies of Roget77 have much in common with what we believe ought to be found in taxonomies for knowledge bases.

---

[2] There has been virtually no emphasis on making ontological assumptions explicit in the literature. Regoczei&Plantinga87 have discussed the importance of ontological matters. Others have recommended making tacit knowledge explicit, especially with respect to rule based systems (e.g. Clancey83). Others have recommend clarifying the semantics of knowledge representations (Brachman85, Woods85). We go considerably further in recommending the explicit specification of all ontological assumptions.

Knowledge bases have characteristics of all three. Most designers of knowledge bases have shown little interest in these distinctions, terminology, or problems associated with these three classical forms of structured knowledge, but part of our approach (due to our background in language-intensive applications of knowledge bases) is to draw inspiration from some of these similarities.

In this paper, we will attempt to elucidate a number of the key issues in ontology design, using the three models cited above as a point of reference. We then will make some recommendations about how to approach questions of *methodology* in particular. The paper is organized as follows: In Section 3 we introduce some of our meta-terminology; in Section 4 we discuss some major issues other than those concerning methodology; Section 5 describes our linguistically-oriented methodology, while Section 6 points out some additional methodology issues that apply general. Section 7 offers concluding remarks.

3 SOME BASIC CONCEPTS AND TERMINOLOGY

First we clarify our use of some key terminology. We use the term *knowledge* to mean domain specific knowledge that experts in the given domain agree on. By *ontology* we mean the very general, usually tacit, views about basic semantic categories (e.g. entities, properties, situations) and their properties that people share. Because ontology is usually tacit, a key problem (on which we will focus) is how to establish ways of uncovering these assumptions and rendering them explicit, for only thus can we hope to reach agreement among and between knowledge base designers and users. Although an initial impulse of many upon first encountering the idea of ontology is to think that everyone has a different ontology or that there are as many ontologies as points of view, we believe that people's ontological views will have much in common once they are made explicit through appropriate analysis and criticism.

Knowledge bases are computer repositories containing domain specific knowledge, employing anything from vague, unconscious ontologies to very precise ones. By an *ontology base* we mean a well defined set of very general top-level *semantic categories* and associated *relations* and *assumptions* that makes the ontology as

explicit as possible. An ontology base, which should precede knowledge acquisition, may be open to modification as knowledge acquisition proceeds, though too much modification would reduce its effectiveness as the key medium for explicating the ontological assumptions behind the knowledge representation, the knowledge itself, and the acquisition process. Thus one adds domain-specific knowledge "below" an ontology base, and sharing and reuse is most enhanced when the ontology base becomes stable.

We suggest that it may be possible to create a *general* ontology base, general enough to be useful in virtually any subject area, i.e. which does not include any knowledge specific to some area. From such a base, *restricted ontologies*, e.g. for medicine or electrical engineering can be constructed. By this we mean an ontology base extended to contain the general concepts for the domain, i.e. that are subcategories of the higher, totally general ontological categories. This is the main way that a general ontology base guides knowledge base construction. It helps to ensure that the knowledge base can be used for various purposes in the same domain, or when coupled to others in different domains that share the same global ontology. Since such ontology bases are as yet not available for reuse, knowledge base developers, including ourselves, are constantly having to "start from scratch", designing and redesigning ontology bases without any agreed upon standards or methodology. These ontology bases usually are constantly being modified as new knowledge is added, causing serious consistency and reworking problems. Thus, one is not able to build effectively on other peoples' work.

Another term that concerns us is "primitive". In general, the term *primitive* usually refers to a semantic category, relation, or term that is used to define or describe others. It may or may not be defined circularly using other primitives. Sometimes, as is the case with the defining vocabulary of the LDOCE, a restricted set of terms or concepts is selected constituting a controlled vocabulary. (LDOCE nevertheless defines each primitive in terms of the others.) While the top level categories that form an ontology may be called primitives by some, we prefer to call them "the top level categories" and reserve the term "primitive" mainly for relations, such as those termed "epistomological" by Brachman.

Two other closely related key terms are *category* and *concept*. We suggest that the former apply to knowledge units in the ontology

base, while the latter apply to all other units (lower in the knowledge base). Thus semantic categories are just high level notions which necessarily are defined in terms of other peer level categories. What *characterizes*[3] a category then is its relationship to other categories, whereas concepts may rely both on the higher concept level, and each other.

A *knowledge unit* is a collection of elementary assertions about one subject. It may correspond in "size" to a paragraph or two in natural language, and often is represented as a frame structure in a computer. A clear assertion regarding the question of definition or characterization we term the *definitional status* of a knowledge unit.

4 MAJOR NONMETHODOLOGICAL ISSUES IN ONTOLOGY

We discuss next some of the major issues other than methodology surrounding the design and use of ontologies. This list is far from exhaustive; space and time restrictions led us to focus on some of the most crucial. In Section 5, we focus on methodological issues.

The question of what should be the most general (top level) semantic categories[4] goes back at least to Aristotle66, who proposed some that today remain very plausable. In the nineteenth century, Roget proposed a very exhaustive set of categories for his thesaurus (Roget77), though their meaning was only implicitly conveyed by listing similar terms in the same category. Among ai researchers, we know of only two proposals for general ontologies that have appeared in books, those of Dahlgren88 and Lenat90., and one that has appeared as a techical report (BatemanEtAl90) In Dahlgren, details of many important categories have not been described. Lenat does attempt to describe informally most top level categories[5]

---

[3] We prefer using the term "characterize" rather than "definition" when talking about the meaning of a category or concept in a knowledge base.

[4] Most concepts are fuzzy, yet to date, the vast majority of knowledge acquisition systems use knowledge representations that are strictly symbolic and non-fuzzy. The recent movement toward connectionist representations is an extreme reaction to this serious limitation. Since it may be many years before connectionism or other approaches to fuzziness are sufficiently developed, some accomodation of fuzziness may be necessary in existing symbolic models. However, our goal is to design the top level categories as crisply as possible.

[5] We plan a comprehensive study of these and the methodology of their exposition.

however, neither is explicit about methodological matters. The BatemanEtAl work derives from the systemic grammar theory of M. A. K . Halliday85, and is the only one to be based on sound linguisitic knowledge. While each category is described, it is difficult to follow without reading Halliday85. Lack of adequate documentation and lack of attention to methodological issues are key issues that need to be resolved. We would like to see each category explained as clearly as possible (admittedly not an easy task), using carefully chosen and consistent terminology, together with the rationale for choosing the category. Ideally, the complete text of each knowledge unit, suitably explained at least in English, should be available.

Here is a list of some of the terms that are commonly used to name the most important categories, as we see it (object level terminology):

concept, object, thing, entity, situation, relation, property, attribute, state, event, process, activity, action, part, whole

Some typical meta-level terminology includes:

type, class, intension, extension, instance, token, individual, referent, predicate, proposition

We look forward to the day when this and similar terminology will be used much more clearly and consistently than at present. We would hope that soon, we will be able to present our proposals for some definitions for such terms.

We do not have space to discuss some other ontologies that have been proposed. Some other representative examples of general ontologies may be found in Wierzbicka85, NirenbergEtAl88, Barwise and Perry83 None of these except Wierzbicka include any attempts at explaining the intended meaning of the categories or the methodology. Examples of restricted ontologies, which include more explanation of categories, are HobbsEtAl88 and Hayes88.

Another key question is whether the notion of *semantic primitives* is useful at all, and if so, what this term should mean. The term is used in various ways; some use it to mean the top level categories, others, to mean primitive binary relations, often associated with semantic network representations, the sense we use here (Woods85, Brachman85, Sowa84). From a linguistic point of view, only a few sets of primitives that might be termed "semantic" have been

proposed, which we cannot review here (Wilks77, Schank&Riesbeck81, Melcuk89). A methodology should make clear whether or not it considers that parts of knowledge units constitute some type of definition or characterization, and whether or not some notion of semantic primitives (either restricted to relations or in the more general sense) is being used. Some researchers believe that it is either hopeless to seek such primitives at this point, or that it may not even be desirable, i.e. that knowledge bases should look more like thesauri than LDOCE-style dictionaries. But only by attempting to find such primitives will we eventually learn if this approach is fruitful.

The knowledge representation chosen also has a major impact on ontological issues. In most cases, some fixed (or often evolving) knowledge representation is prescribed (e.g. CYC and KBMT89 (Goodman89)); in others it is left open-ended. The degree to which the limitations of the knowledge representations affect ontological assumptions is, we suspect, often underestimated. For example, users of frame-based representations may make the assumption, consciously or not, that all relations (represented as slots) are binary and have a "directionality", i.e. a "from" and "to" argument that conceptually may not make sense in that both arguments should be equal partners. Another undesirable commitment may be that all attributes must have a value (e.g. the value of "shape" = "square"; the value of "sex" = "male"). A third common cause of confusion comes from using the same "frame" syntax to represent both "class" and "instance" knowledge. The meaning of the two is quite different, but the frame syntax may look identical. We believe that such decisions should only be made when it is realized a) that they may be an artifact of the knowledge representation, b) what are the alternatives, c) what are the consequences.

A final important point is that an ontology base should be able to assist in a semantic classification of closely synonymous terms in not only one but *various* natural languages and, in that sense, should form the basis of an intertranslatable thesaurus. Hence it is important to establish standards and methods for reaching agreement upon a set of ontological categories, all the more so if we concern ourselves (as we should) with translatability.

Our Maxim:

Without a clear understanding of the ontology base, any knowledge base is open to misunderstanding and improper use.

5. TOWARDS A LINGUISTICALLY BASED METHODOLOGY FOR DESIGNING ONTOLOGIES: AN EXAMPLE

5.1 Introduction

We have been trying to simultaneously design a general-purpose ontology base and identify the underlying issues like those we raised above. We have focussed on trying to introduce as much knowledge as possible from the linguistics and cognitive science literature[6] into the process, since we believe it can offer the most assistance in making ontological choices. Thus we have two goals:

a) to produce an ontology base that has a firm basis in existing ontological and linguistic principles, that is general and useful, that is well documented, and that has been validated (see 6.2) by a significant set of knowledgable users or potential users. We view this as a several year project.

b) to report in particular on our approach to the methodological issues.

We are not yet ready to discuss the former, and this paper is a first attempt at the latter. In describing our methodological approach, we will refer to only one representative branch of the ontology, the 'attributes'.

One of the difficult ontological categories is the so-called *attributes*, also often termed "properties", "qualities", or "features". To give the reader a quick idea of where attributes fit in our ontology, we have three main top level categories: entities, situations, and properties.

We have two subcategories of property, the *relations* and the *attributes*. Relations involve two or more things on a roughly equal status, whereas attributes involve primarily only one, which they

---

[6] Shaw and Woodward89 emphasize the importance of drawing upon the cognitive science literature for clarifing the notion of a "conceptual model", which is very similar to our notion of "ontology".

supply information about. Syntactically they originate mainly as adjectives and adverbs. We cannot go into here our detailed analysis of attributes, but we have tentatively identified a half dozen or so subcategories and associated primitives.

We discuss next some of the methodological issues we encountered in our analysis and our response to them.

5.2 Consult the Linguistic and Cognitive Science Literature

We had great difficulty in trying to devise a conceptual structure for attributes, and in finding reports of such attempts by others on which to build. We found virtually nothing useful in the ai literature. Most ontologies in existence in ai have usually been built for very practical purposes. The designers probably did not make any deliberate attempt to incorporate concepts from disciplines like linguistics, psychology, and philosophy, relying instead on what they might find in the ai literature and on their own intuitions about language. This is understandable given that

* people may not feel that these areas are relevant (e.g. there is relatively little mention of the importance of linguistics in the knowledge acquisition literature)
* it is not clear where and how to incorporate such ideas
* the needed concepts are still very vague and controversial (e.g. reading the philosophy literature can be a frustrating exercise for non-philosophers, yet it is there that the most fundamental discussions of ontological problems are to be found)
* most ai workers do not know this literature, which is often hard to find, let alone understand. They may not even know someone who could help them.

In our attempts to decide on semantic categories and in particular to discover semantic structure for attributes, we consulted the linguistic, cognitive science, and philosophy literature. For some major semantic categories, like situations, events, or processes, we have collected many useful articles. However there has been relatively little work on attributes, but we have discovered a few (e.g Warren84). Our point is that the linguistic literature was virtually our only source of ideas for ontological analysis, since, in our view, ontology is intimately associated with language. This reinforced our

view that a language based methodology is a promising approach for establishing an ontological base.

5.3 Standardize Terminology and Naming Conventions

The problem of finding and using consistent and rational terminology, both at the meta and the subject matter levels, is particularly serious in ontological design, since the very essence of such research is to find clear, generally accepted concepts and associated terminology by which to communicate knowledge to others. We recommend trying to maintain and follow glossaries of all technical terminology as carefully as possible.

The *naming problem* is that often there are no criteria for choosing names of knowledge units or their components (e.g. frames and slots). We feel there should be a global dictionary of all names used and that this should be an integral component in any knowledge base system. It could be coupled to a machine-readable dictionary and could enforce certain syntactic conventions. We have established some simple syntactic criteria for the choice of slot and frame names in our systems; we do not allow long phrases that contain "buried knowledge".

5.4 Take a Clear Position on Definitional Status and Primitives

Above, we pointed out the questions of definitional status and semantic primitives, and we return now to these problems. English does not have a good term for the former notion: "the knowledge unit is a _____ for the concept". However the term "definition" is often used by knowledge base personnel, and even lexicographers sometimes refer to the entries found in typical dictionaries as definitions. We believe that this is a specialized meaning of the term, and we prefer the looser term "characterization".

Often a knowledge unit will have many parts (e.g. slots) and some of these might constitute an actual "definition" (if there is one), the rest constituting knowledge that should be termed "encyclopedic", the term usually used by lexicographers. Hence knowledge units (that correspond to a type) should clearly distinguish

* parts that constitute an actual definition or characterization, if there is one
* if appropriate, necessary and typical conditions that instances have
* same for sufficient or typical conditions that imply something is an instance
* any other knowledge that is not about instances

Turning to the semantic primitive question, we may consider two extremes:

1 there is no set of prescribed "primitives", i.e. a knowledge unit may use any combination of already defined concepts and undefined ones in making a new concept characterization;

2 a set of primitives is given, and all characterizations must either use it directly (e.g. like LDOCE), or must use others which can be reduced to the primitives (e.g. like in mathematics).

We suggest that at the moment, neither extreme is desirable. The former leaves the door open to an undisciplined proliferation of terms, both undefined and badly defined (e.g. circularly). The latter will not be possible until much more research has been done on selecting a good set of categories and primitives, assuming one subscribes to this approach. Hence a middle ground should be sought: a partial, initial set of semantic categories and primitives is given, which is under gradual evolution. Knowledge enterers are asked to try to use them as much as possible, i.e. by introducing new concepts only as specializations of top level categories, and to note carefully every place they feel there is none sufficient for some need, so that "holes" in the set can be discovered and covered.

We suspect that the bulk of knowledge base development today follows 1. We believe 2 may be a better alternative, if it becomes possible; it is too soon to tell. It will probably draw much of its inspiration from research in dictionary and thesaurus construction, which is still itself in a "primitive" state. We suggest the two endeavours of finding good primitives for knowledge base construction and good primitives for dictionary construction are two sides of the same problem. Workers in these two fields, who have not been aware of each other in the past, are beginning to cooperate (e.g. Skuce and Meyer89)

5.5 Use Well Defined Corpora

Although this might seem obvious, we believe that some knowledge bases are constructed from a haphazardly selected collection of knowledge sources, and the contribution of each is far from clear. This makes it virtually impossible to predict what knowledge to expect in it, or to verify any knowledge unit. A knowledge base should have associated with it a clear statement of the sources used. Our methodology has focussed on textual sources, which we can cite.

When interested only in a narrow subdomain (e.g. the IBM XT, used in the KBMT89 project), one may define the coverage by collecting all terminology from one or a few well-prepared documents on the specific subject as a guide to the set of concepts that must be captured. Some automated method for accumulating terminology sets should be used (e.g. Szpakowicz90).

However, if one is interested in creating general purpose ontologies (as we currently are), there are two well-defined sources for comprehensive word sets that seem useful:

* a general corpus (e.g. the "Brown" corpus for English (Kucera&Francis82))
* the "controlled vocabulary" used in the LDOCE[7]. For example, in our current analysis of attributes, we are using the adjective set from Johansson75.

5.6 Dealing with Polysemy

Many, if not most, of the words used to name either knowledge units or relations (e.g. slots) in typical knowledge bases are polysemous, without any account of which meaning is intended being made. The creators may not be aware of this, since one meaning is usually fixed in their minds. A similar and surprising example is that much of the defining vocabulary in the LDOCE is polysemous (the reader is expected to disambiguate) (Wilks90). Part of our approach involves identifying such terms and clarifying which meaning is intended, to ensure that they are used consistently. There is no way we know of

---

[7] Yorick Wilks90 and his associates are currently working to improve this set and to make it more suitable for use in natural language research.

for a machine to identify inconsistent sense uses. To help deal with this serious problem, we would suggest:

* choosing terms with as little polysemy as possible (an online dictionary would help)
* replacing polysemous contractions with full terms that distinguish the meanings[8]
* attaching sense subscripts where unavoidable (these could appear only on request)
* having selectional restrictions and other "type checking" mechanisms built into the system

For the other cases, a system that could pop up the various meanings during review by knowledge validators would permit checking that every use conformed to the intended meaning. This would be laborious, to be sure, but careful knowledge validation unavoidably is.

5.7 Deciding Upon a New Category or Concept

There are two extremes, but most knowledge entering falls in the middle. Starting from "scratch", in what we call ontological design, someone must create an ontology base, i.e. decide on all the most basic categories and relations (assuming the knowledge representation issue is settled.) This is extremely difficult and crucial: all else depends on it. Only highly experienced knowledge workers should attempt it, nevertheless equally experienced workers will probably come up with quite different ontologies, unless they share the same environment or beliefs (and this does not guarantee that they won't!) We cannot give guaranteed sources in which one is sure to get ideas for these primitive categories and relations, since it is a highly creative endeavour, but here are some we use:

* read the linguistics, philosophy, or psychology literature
* look at other people's knowledge bases/ontologies
* take some domain and try to generalize from it

---

[8] A recent example encountered by one of the authors: the technical term "actor" was causing much confusion, since it turned out to be a contraction for either "actor specification", "actor reference", or "actor incarnation". Another example: a knowledge base in which units for "garbage", "trash", and "rubbish" had been created without any apparent connections being made between them.

* select a comprehensive set of terms of a particular type (e.g., our adjectives) from a well specified corpus
* introspect

We suspect that most ontologies in use today evolved by some unclear combination of the above plus other methods. Probably use of the non-ai literature is not common, except perhaps the use of the linguistic literature by those specifically interested in natural language applications, as we are. As we have stated, we have yet to read any detailed account by a creator of an ontology of how it was done. We are currently documenting our work on attributes, partially to clarify by example how we would like to see it done.

At the other extreme, a knowledge enterer may simply fill in "values" or "instances" of predefined categories and properties. This is analogous to filling in the spaces on a typical office form (though some forms can be puzzling). Hence this can be done by persons with less training, and the system can do a lot of constraint checking. We are not concerned with this case.

The great bulk of knowledge entering however falls somewhere in the middle: the knowledge enterer inherits some existing conceptual structure, both ontological and at lower levels, but still must make some difficult decisions on new categories and properties; the "higher" these are, the more "ontological" and difficult they may be (if for no other reason that there may be a larger body of concepts "beneath" that will be affected). Sometimes some of the existing ones need modification. We would recommend that there be various "levels" at which only certain types of modifications are allowed, controlled by designated authorities.

It is in these intermediate levels that the machine could potentially assist the user most (we assume that with the "ontology level" decisions, little or no machine assistance is feasible at present). We return to this question below when we discuss "bottom up" knowledge acquisition.

* select a comprehensive set of terms of a particular type (e.g., our adjectives) from a well specified corpus
* introspect

We suspect that most ontologies in use today evolved by some unclear combination of the above plus other methods. Probably use of the non-ai literature is not common, except perhaps the use of the linguistic literature by those specifically interested in natural language applications, as we are. As we have stated, we have yet to read any detailed account by a creator of an ontology of how it was done. We are currently documenting our work on attributes, partially to clarify by example how we would like to see it done.

At the other extreme, a knowledge enterer may simply fill in "values" or "instances" of predefined categories and properties. This is analogous to filling in the spaces on a typical office form (though some forms can be puzzling). Hence this can be done by persons with less training, and the system can do a lot of constraint checking. We are not concerned with this case.

The great bulk of knowledge entering however falls somewhere in the middle: the knowledge enterer inherits some existing conceptual structure, both ontological and at lower levels, but still must make some difficult decisions on new categories and properties; the "higher" these are, the more "ontological" and difficult they may be (if for no other reason that there may be a larger body of concepts "beneath" that will be affected). Sometimes some of the existing ones need modification. We would recommend that there be various "levels" at which only certain types of modifications are allowed, controlled by designated authorities.

It is in these intermediate levels that the machine could potentially assist the user most (we assume that with the "ontology level" decisions, little or no machine assistance is feasible at present). We return to this question below when we discuss "bottom up" knowledge acquisition.

5.8 Importance of Language Independence

As we have pointed out, it appears that most knowledge base creators have paid little or no attention to lexical matters. One example we have pointed out is the casual use of words in various senses, without distinguishing these senses. A more serious problem, we believe, is dependance on a particular language for meaning. For example, in English-speaking countries, English words and phrases normally are used as names for knowledge units or relations. These may have no natural translation into other languages: a Japanese person should not have to become expert in English to reuse a knowledge base that originated in English. Hence if we are to move toward the goal of widespread use of knowledge bases, care must be taken that this set is not just a good one for, say, English. Thus knowledge bases can become accessible to speakers of other languages; ultimately these would be easily accessible though automatic translation.

To do this, we must ensure that at least the ontology base uses, as much as possible, a truly "universal" set of categories, i.e. ones that have closely synonymous correspondences (i.e. terms for them) in at least a number of major languages. Since, as we have noted, the problem of finding an ontology base for even English is far from being solved, it will be somewhat longer before any is generally accepted as being universal for, say, the ten most common natural languages. However we would suggest that now is the time to begin thinking seriously about this. For example, as components of ontologies are proposed, they could be at least methodically checked by translators for close correspondents in several other major languages (failing to find a correspondent in language X would cast doubt on the primitiveness of a given concept; failing to find it in several languages would certainly rule it out[9]). A somewhat related paradigm has been adopted in some machine translation systems: only a controlled vocabulary may be used in the source documents. By this means, fully automatic and correct translation will be more likely to be attainable within the near future.

---

[9] For example, Melcuk says that 'cause' is not realized in Russian (Melcuk89), and hence he is doubtful of its status as a "universal semantic primitive". We also suspect that the notion of "attribute" may be very problematic

6 GENERAL METHODOLOGICAL ISSUES

In this section, we describe some additional points that are as relevant to knowledge base construction in general as to ontology, and that apply to any approach, i.e. they are not specific to our linguistic approach.

6.1 Building Knowledge Bases Bottom-up vs Top-down

In our experience, knowledge entry tends to procede in a mixture of several "directions", for want of a better word:

top-down: one enters new concepts as specializations of those already entered. This is probably the easiest, and is the only methodology supported by object-oriented programming systems. Many knowledge acquisition tools do not provide any facilities that encourage any other "direction", making it difficult to go "against the grain". But we find that frequently, one wants to go "up", i.e. to generalize some concept already entered.

middle-out: one enters a concept A that is a specialization of higher level ones, B and C say, but then decides that there is a more general concept than A that is needed. It may be "under" B or C, closely related to them or, in the extreme, completely separate from them, indicating a large "hole" in the knowledge base.

bottom-up: the most specific examples (possibly actual "instances") of knowledge are entered first, and it is desired to generalize these, i.e. to discover what classes they belong to based on shared properties, in other words, to erect a hierarchy of more general concepts that subsume them. This mode of knowledge entry is not supported at all by many tools; it is however, the express purpose of a few knowledge acquisition systems to allow grouping and classification hierarchies to be developed inductively from examples (e.g. KSS0 RReference needed)[10]. Indeed, it is the raison d'etre of so-called inductive learning programs to create generalizations from examples. The machine learning literature, which tends to concentrate on

---

[10] We do not believe such a strategy would work for inducing the top level categories. In fact, a well defined set of such categories is probably a prerequisite for fruitful induction.

automatic methods for doing this, is beginning to report on some interactive generalization techniques (RReference needed). We have in mind a hybrid form of learning system, in which the machine proposes possible generalizations of categories, or new properties, or shows all concepts that are "similar" or "related" to a given one, with the human passing judgement on the proposal, accepting it either outright, with modification, or not at all. Some rudimentary examples of such a mechanism can be found today in some current knowledge acquisition tools[11]

We conclude that all three directions are essential, and that the next generation of tools should support them equally.

6.2 Validation Techniques

In parallel with the situation in software engineering, an unverified knowledge base is as desirable as an untested program. We have not seen any significant knowledge base described where there was any discussion of whether it had been in any way verified, let alone how[12]. How was the knowledge base verified? Against what? Who did it? What procedures were used for verification? Does the system assist in the verification process in any way, at least in the mundane bookkeeping, e.g. flagging those knowledge units that are, e.g., a) to be checked, b) being reviewed by a team, c) passed by the team?

The problem of knowledge base verification is not quite the same as for software. In the latter case, some documented set of requirements and specifications are first *validated*, i.e. checked by those requiring the software to ensure that their wishes have been faithfully described. When the system is created it may then be *verified*, by an independent critic, that it meets the requirements and specifications. In the case of a knowledge base, there are two cases:

1. There exists some natural language document that has served as the knowledge source. Verification in this case would mean that persons (who obviously should not be part of the knowledge entry team) would study each knowledge unit, and agree that their understanding of them was consistent with the document.

---

[11] e.g. the generalization mechanism in CODE (SkuceEtAl89), and CYC's "analogy" or "copy and edit" mechanism (Lenat90).

[12] E.g., Lenat and Guha say nothing about this.

2. There is no document: the knowledge base itself is the only source of the knowledge other than the minds of those who created it. This case is even more difficult. Some form of "structured walkthoughs" (to use the software engineering term) seems to be the only approach at the moment, and the term "validation" might be more appropriate, though we do not see a need to use both terms. Teams of experts, again preferably not those who created the knowledge base, must examine it knowledge unit by knowledge unit and agree that what each says is correct. Where they cannot come to agreement, some indication should be attached to the offending parts, e.g. that the fact expressed by slot ABC is accepted only by 3 out of 5, or some such warning. Thus users of ABC can draw their own conclusions before relying on it, or can research it further.

We have some experience with the latter case. The CODE system has facilities for this purpose, which were used heavily in a knowledge acquisition experiment at Bell Northern Research in the fall of 1989.(Skuce89). Each knowledge unit was created by an assigned member of a team of four, in close collaboration with DS, who acted as a kind of coordinator or "knowledge base administrator". When the enterer felt the unit was ready for scrutiny by the other members of the team, he changed its "status" flag from "in progress" to "under review". At regular intervals, the team of four would spend a whole day critiquing each other's units, either cleaning them up on the spot or sending them back to the drawing board. The system assisted by, for example, quickly locating all knowledge units "done by Smith since last week and not accepted". We found this method of peer review (a new meaning for "peer review"?) effective in controlling the evolution of the knowledge base, but it is only a beginning.

6.3 Standards and Documentation

There has recently begun a standardization effort to draw up some agreements on knowledge bases (Neches90). The early documents circulating by email have not yet shown much concern for the types of issues we have brought up here. The concern has been more toward standardizing the knowledge representation, rather than the higher-level problem of what should be the ontology base.

Standards for software documentation have been developed by several agencies like IEEE84. We would like to see analogous standards developed for documenting knowledge bases and most important, ontologies. Could there in fact be some day an ISO ontology base?

7 CONCLUSIONS

It has taken the field of software engineering some fifty years to evolve from the first hand-coding of the machines of the 'fourties to today, where, at least in textbooks, there are some methodologies that one can look up and follow. However, knowledge engineering today, and the problem of ontological design in particular, is still in an embryonic state. We have attempted in this paper at least to draw attention to some of the problematic issues, which for those caught up in the difficulties and deadlines of knowledge acquisition, could easily go unnoticed, as we ourselves have found. Our objectives were twofold with regard to ontological issues:

1 to bring into the open many often unstated or unconscious issues so that knowledge base designers may ask themselves: "what is our position regarding ___?", or "what does our system do with _____?". Eventually, a generally accepted checklist of such issues could be available, and part of the documentation for a knowledge base could be an explicit discussion of each issue.

2 to describe some of the linguistically motivated methodology we have been experimenting with for ontological design.

Since our top-level goal is to open these issues for discussion and to encourage dialog that could lead to consensus, we welcome any comments or suggestions.

ACKNOWLEDGEMENTS

Funding from The Center for Machine Translation at Carnegie Mellon University (while DS was there on sabbatical) contributed to the writing of this paper. Valuable suggestions were provided by Jaime Carbonell.

REFERENCES

Aristotle. Aristotle's Categories and De Interpretatione, translated by J.L. Ackrill, Oxford University Press, 1966.

Barwise, J., and Perry, J. Situations and Attitudes, MIT Press, Cambridge, 1983.

Bateman, John, Kasper, R., Moore, J., and Whitney, R. A General Organization of Knowledge for Natural Language Processing: the Pennam Upper Model. unnumbered technical paper, Information Sciences Institute, March, 1990.

Brachman, R. On the Epistomological Status of Semantic Networks. In: Readings in Knowledge Representation, (Brachman, R. and Levesque, H., eds), Morgan Kaufman, San Mateo, 1985.

Clancey, W. The Epistomology of a Rule-based Expert System- A Framework for Explanation. Artificial Intelligence, 29, 215-251, 1983R

Dahlgren, K. Naive Semantics for Natural Language Understanding. Kluwer, Boston, 1988.

Halliday, M. A. K. Introduction to Functional Grammar. Edward Arnold, London, 1985.

Hayes, P. Naive Physics I. Ontology for Liquids. In: Formal Theories of Commonsense Reasoning. (Hobbs, J and Moore, R., eds) Ablex, Norwood, 1988.

Hobbs, J., Croft, W., Davies, T., Edwards, D., and Laws, K. The TACITUS Commonsense Knowledge Base. (draft) SRI International, 1988.

IEEE Guide to Software Requirements Specifications. IEEE, New York, 1984

Johansson, S., Some Aspects of the Vocabulary of Learned and Scientific English, Acta Universitatis Gothoburgensis, Goteburg Sweden, 1975,

Goodman, K. (ed) Special Issue on Knowledge-based Machine Translation. Machine Translation, 4, 1 and 2, 1989.

Kucera, H., and Francis, W. Frequency Analysis of English Usage. Houghton Mifflin, Boston, 1982.

Lenat, D. and Guha, R. Building Large Knowledge-Based Systems, Addison-Wesley,1990.

Longman Dictionary of Contemporary English. Longman, Harlow, 1978.

Mel'cuk, I. Semantic Primitives from the Viewpoint of the Meaning-Text Linguistic Theory. Quaderni di Semantica X, 1, 65-101, 1989.

Monarch, I. and S. Nirenburg, The Role of Ontology in Concept Acquisition for Knowledge-Based Systems, Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Reading: University of Reading, 1987b.

Neches, R. (co-chair) Workshop on Standards for Knowledge Representation, Santa Barbara, March, 1990 (not published).

Nirenburg, S., Monarch, I., Kaufmann, T., Nirenburg, I. and Carbonell, J., "Acquisition of Very Large Knowledge Bases: Methodology, Tools and Applications", Third Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1988.

Regoczei, S., and Plantinga, E. Creating the Domain of Discourse: Ontology and Inventory. Int. J. Man-Mach. Studies, 27, 3, 235-250, 1987.

Roget, P.M., Roget's International Thesaurus, revised by R.L. Chapman, Harper and Row, New York, fourth edition, 1977.

Schank, R., and Riesbeck, C. Inside Computer Understanding, Lawrence Erlbaum, Hillsdale, 1981.

Shaw, M, and Woodward, B. Mental Models in the Knowledge Acquisition Process. Proc. 4th Knowledge Acquisition for Knowledge Based Systems Workshop, Banff, 1989.

Skuce D., and Meyer, I. Concept Analysis and Terminology: A Knowledge-Based Approach to Documentation. To appear in Proceedings of the Thirteenth International Conference on Computational Linguistics (COLING90)

Skuce, D and Meyer, I Computer-Assisted Conceptual Analysis: An Essential Component of a Terminologist's Workstation. To appear in Second International Congress on Terminology and Knowledge Engineering Applications, 1990.

Skuce, D. Conceptual Analysis: A Generic Environment and an Example from Software Engineering. Dept. of Computer Science Tech. Rpt. 89-32, 1989.

Skuce, D., Shenkang, W., and Beauville, Y. A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis. Proc. 4th Knowledge Acquisition for Knowledge Based Systems Workshop, Banff, 1989.

Sowa, J. Conceptual Structures: Information Processing in Mind and Machine. Addison Wesley, Reading, MA, 1984.

Szpakowicz, S. "Semi-automatic Acquisition of Conceptual Structure from Technical Texts". Int. J. Man-Machine Studies, 32, 1990.

Warren, B., Classifying Adjectives, Acta Universitatis Gothoburgensis, Goteburg Sweden, 1984.

Wierzbicka, A. Lexicography and Conceptual Analysis. Karoma Publishers Inc., Ann Arbor, 1985.

Wilks, Y. Good and Bad Arguments about Semantic Primitives. Communication and Cognition, 10, 182-221, 1977.

Wilks, Y., Fass, D., Chen-ming, G., McDonald, J., Plate, T., and Slator, B. Providing Machine Tractable Dictionary Tools. To appear in Machine Translation, 1990.

Woods, W. What's in a Link? Foundations for Semantic Networks. In: Readings in Knowledge Representation, (Brachman, R. and Levesque, H., eds), Morgan Kaufman, San Mateo, 1985.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

ACQUIRING STRATEGIC KNOWLEDGE IN ADVERSARIAL DOMAINS

Steven Walczak
Computer and Information Sciences Department
Room 301, CSE Building
University of Florida
Gainesville, FL 32611

ABSTRACT

Acquisition of knowledge typically involves the cooperation of an expert in the domain field. Experts cannot be expected to cooperate with a knowledge engineer or knowledge acquisition tool in adversarial domains, since such cooperation might adversarly affect the survival of the expert. This paper presents a method for acquiring knowledge from adversaries by direct observation of an adversary's performance in the domain. Direct observation is available in most adversarial domains through historical records of the adversary's previous performances. An implementation of the described method has been accomplished for the domain of chess and the results produced from the acquired knowledge are discussed.

INTRODUCTION

Adversarial domains include military planning, economic planning, law enforcement, athletics, and board games. We limit the consideration of adversarial domains to those which have a zero-sum outcome and are not directly influenced by luck to enable qualitative acquisition of knowledge. Computer systems operating in adversarial domains require knolwedge about the domain and their adversary to make optimal decisions. Gruber (1988) and others (Langley, 1985) have already investigated the acquisition of strategic knowledge from cooperative experts. However, adversarial domains do not lend themselves to knowledge acquisition from an expert since the expert may well be an adversary of the computer in the future. The domain expert will try to mislead the knowledge acquisition system about his true intentions in the domain. In this paper, we demonstrate a method for acquiring knowledge about the style and intentions of an adversary from direct observation which does not require a cooperative effort from the adversary.

Game theory bas long been used for decision making in tactical and strategic situations (Haywood, 1954; Thomas, 1984; Von Neumann and Morgenstern, 1953) as well as resource allocation problems (El-Fattah and Foulard, 1978). Game theory however, provides two different methods for evaluating domain information. The U.S. military uses the method of evaluating opponent capabilities, similar to producing an exhaustive searcb of a game tree for a chess problem, instead of the method of evaluating an opponent's intentions which is analogous to only evaluating the brancbes of a game tree that the adversary is actually considering. The ability to predict the choices an adversary will make in specific domain

situations affords a significant strategic and tactical advantage. For example, athletic coaches attempt to utilize such information by relying on scouting reports and films of their adversaries (Aliston and Weiskopf, 1984; Riggins and Winter, 1984).

The ability to accurately predict an adversary's decisions in tactical and strategic situations increases the speed, by eliminating fruitless branches in the game tree, and accuracy of search algorithms used to emulate human experts in strategic and tactical domains. Current efforts to heuristically incorporate the goals and beliefs of an adversary into strategic planning programs, such as POLITICS (Carbonell, 1981) and MARK II (Davis, 1988), rely on the programmer's beliefs about the goals and intentions of specific adversaries. This approach is hindered by the tendency of humans to view enemies as clones of themselves which produces a tactical blindness (Dunnigan, 1982). The use of game theory in selecting strategic plans also suffers from the inability of planners to accurately capture the true values that an adversary associates with specific objectives. Historic examples of the adverse outcome made possible by ascribing our own beliefs onto an adversary are the losses suffered by the U.S. military during the Japanese attack on Pearl Harbor and the 1968 Tet Offensive in Vietnam.

The knowledge acquisition method we describe has been implemented for the domain of chess in the program IAM, Inductive Adversary Modeler. Chess was chosen due to the strategic and tactical basis of the game and a complexity level which prohibits a solution from being found by an exhaustive search.

PSYCHOLOGICAL BASIS FOR ADVERSARY MODELING

The acquisition of the playing style and strategic and tactical tendencies of an adversary from the direct observation of an adversary's performance in the domain results from the following three part hypothesis. In each part of the hypothesis, the general hypothesis is stated first with its corresponding translation for the domain of chess following.

1. Relevant plans and patterns are learned by the adversary from practical experience in the domain. Chess masters acquire several thousand patterns of pieces that are used to evaluate chess board positions.
2. The strategic actions of an adversary will be repeated as long as the actions produce positive feedback. Chess masters will continue to use a specific strategy while winning games and tournaments.
3. An adversary will tend to reduce the complexity of a situation to enable a more precise strategic evaluation of the situation by using the plans and patterns already acquired in the domain. Chess masters will move board positions that do not contain familiar patterns into positions that do contain familiar piece patterns.

The first part of our hypothesis is a direct result from the research of Chase and Simon (1973; 1988) which claims that human chess masters store from 10,000 to 100,000 patterns of pieces. Chess novices use slow deductive reasoning, similar to an exhaustive breadth-first search, to solve chess problems, but chess masters use fast perceptual processing. The chunking of several pieces into a familiar pattern produces a cognitive economy in evaluating complex board positions, thus allowing the chess masters to select better moves.

The second part of our hypothesis is supported by the Reinforcement Theory of psychology and is an extension of the results of various psychological experiments. The psychological experiments, which include the water jug problem (Anderson, 1980) and the 1023-choice task (Laird et al., 1986), indicate that human problem solvers continue to use learned algorithms for solving problems even when more efficient algorithms would solve the same problem. A practical example of the repetition of strategic and tactical decisions in chess is provided by former World Chess Champion Bobby Fischer who consistently used the PK4, pawn to king four or e4 in algebraic notation, opening in tournament level play.

While the final part of our hypothesis is intuitively appealing, direct support comes from the research efforts of Horgan et al. (1989) and Saariluoma (1984) which both demonstrate the use of chunking and analogy to similar positions by chess masters in solving chess problems. The quantity and quality of chunks used by various chess players results in qualitative differences in playing ability. Chase and Simon's (1973) research indicates that the large collection of patterns or chunks known by chess masters permits them to analyze positions of a game in greater detail than novices who do not utilize chunks of pieces to organize relevant domain knolwedge.

IAM acquires the chunks that are demonstrated by a chess adversary in previous tournament games. The chunks which are acquired are then used to create a model of the adversary's evaluation criteria for the domain and playing style. Acquiring an adversary's evaluation criteria enables us to perceive a domain situation the same as our adversary. We use our new perception of a domain situation to predict the likely move choices the adversary will make in an analogous situation. Knowledge about an adversary's playing style can be used to disrupt the playing ability of that opponent (Holding, 1985). Increases in chess skill are reflected by better predictions of the move an opponent will make in a specific situation.

MODELING ADVERSARY TACTICS AND STRATEGY

As stated in the previous section, IAM analyzes the previous games of an adversary to acquire knowledge about the chunks which are known and used by an adversary. Tournament games are used to limit the amount of misinformation that an adversary is likely to attempt to give to IAM, since the adversary's position in the tournament outcome is at stake. Only games which have resulted in a win or loss for the adversary are studied by IAM. The strict avoidance of drawn games helps to prevent poor quality information. Drawn chess games are frequently influenced by factors that do not impact the strategic intentions or playing style of an adversary. Examples of some non-strategic factors influencing drawn games are: the health or alertness of the adversary which may lead him to draw an early game so that rest can be obtained prior to future games in the tournament and the outcome of the tournament may already be decided thus, making the final game a waste of time for both participants.

Studying the past performance of an adversary to acquire knowledge about that adversary's playing style is commonly performed by master chess players, including the former World Champions Alekhine, Tal, and Botvinnik (Horowitz, 1973; Schonberg, 1973). Berliner (1988) has remarked that human chess competitors collect and study the records of the games played by HITECH so that they may be prepared for a possible future confrontation.

The U.S. military also advocates the acquisition of detailed knowledge about an adversary (Ryan, 1988) and has implemented training programs and facilities which simulate the prefferred tactics of a potential adversary (Robbins, 1988) to train personnel. IAM is simulating this trait of human competitors, the studying of one's opponent, in adversarial domains.

Chunk Composition

Chase and Simon (1973) analyzed the factors that are typically used by chess masters when reconstructing board positions to reveal the basis for chunk creation in human chess players. The most relevant factors found were spatial proximity and likeness of color of the playing pieces. These two factors are natural applications of the Gestalt psychology principles of organization, proximity and similarity. Chunks that are detected by IAM contain a minimum of two pieces, otherwise no cognitive economy is afforded, that are horizontally or vertically adjacent and are all the same color. The limitation of the proximal aspect of chunks to the two dimensional, horizontal and vertical, case follows from the research of Church and Church (Holding, 1985) which revealed that chess players have much greater difficulty processing information from diagonals. We use the Gestalt organization principle of meaningfulness, or a defending relationship from Chase and Simon's study, to include pieces that are capable of moving or defending along the diagonals. Pieces which are adjacent along diagonals to a chunk and which can move along the diagonal to defend a piece within a chunk are included in that chunk. This means that a rook or knight which is diagonally behind a pawn will not be chunked with that pawn. A graphic illustration of the chunks formed for the white player from a specific board position from a 1951 World Championship game between Botvinnik and Bronstein is shown in Figure 1.

Chunks are currently limited to a sixteen square area. This size limitation tries to ensure that the pieces contained in a chunk are relevant to each other and form a strategic or tactical unit. Further discussion of the trade-offs involved with the size of chunks is presented by Campbell (1988). The size limitation also models the hierarchical composition of more complex chunks from a group of smaller chunks performed by human chess masters. The maximum square area of chunks is a modifiable feature of IAM and is only constrained by the size of the game board.

In addition to chunks which are too large, chunks which have greater than half of their pieces still in their starting positions of the game are also discarded. Pieces that have never been moved, in the game of chess, do not reveal information about the strategic plans of an adversary. Thus, the decision to eliminate these chunks from the knowledge acquisition process.

The chunking operation can be extended to other domains. Obvious extensions exist for board games such as Go, Othello, and checkers. To determine chunks for military strategic planning applications, a board representation or map with unit counters to identify the appropriate resources must be instantiated. Hexagonal maps are already commonly used in war games. Identification of unit movement capabilities must also be available to the chunk acquisition procedure to ensure the proximal and meaningfulness principles of chunk construction.

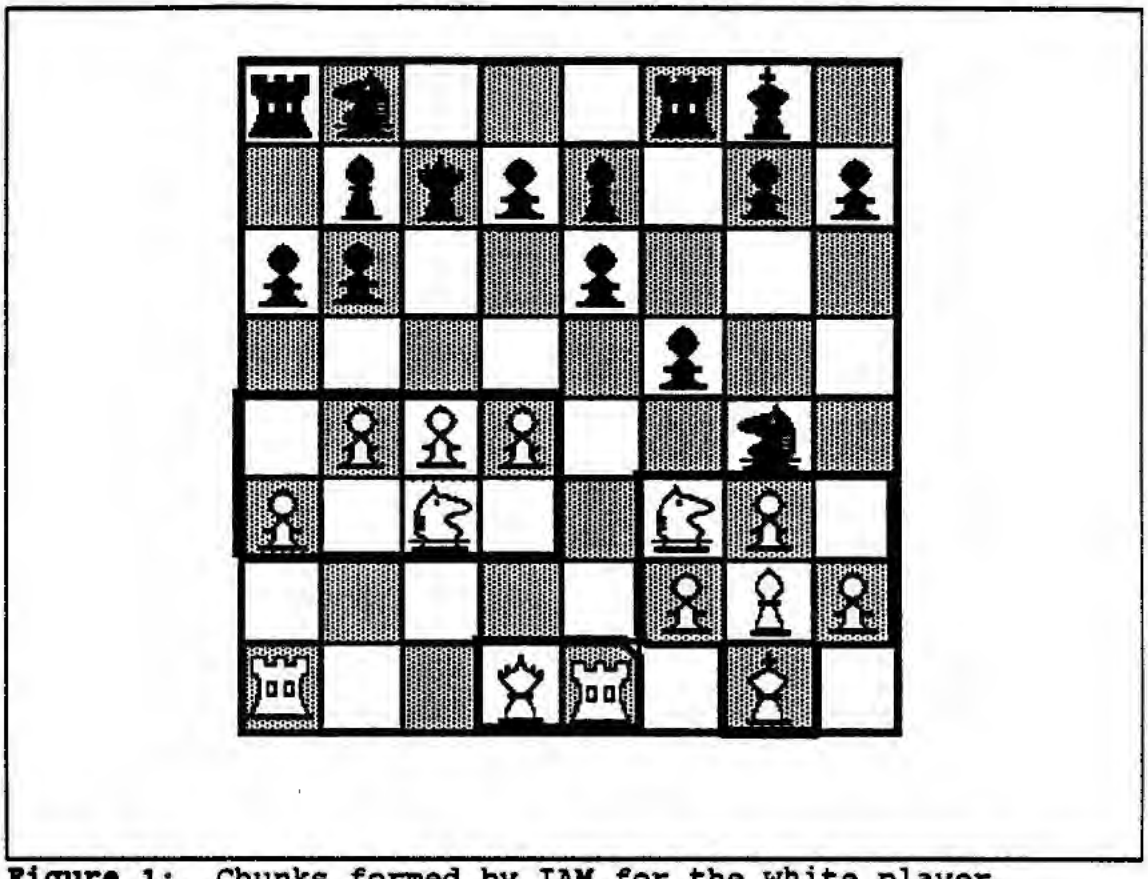


Figure 1: Chunks formed by IAM for the white player.

Holding (1985) and chess grandmaster Nunn (Nunn and Griffiths, 1987) have both noted that in addition to visual processing, chess masters also make use of verbal or textual processing. Nunn states that verbal memory is primarily used to remember opening combinations, similar to the collection of book openings currently used by chess programs, and endgame positions. To augment the knowledge acquired by IAM, the actual moves of the game are also considered to be patterns or chunks. The opening sequences of each game are collected and statistically analyzed by IAM to improve its knowledge during a time of the game when few geometric chunks of pieces exist. This part of the knowledge acquisition process records the opening sequences exhibited by an adversary. This information can then be used to select opening lines of play that are unfamiliar to the adversary and thus gain a strategic advantage.

Chunk Acquisition

Due to the importance of visual processing to chess experts, as demonstrated by Chase and Simon, the computer vision technique of performing convolutions on images to improve their resolution (Ritter et al., 1988) is used to acquire the geometric chunks. The game board serves as an analog to the image being processed by the convolution algorithm and each square of the game board is analogous to a pixel in the image. IAM maintains an

internal representation of the game board and the respective piece locations for use by the convolution algorithm. The board is searched after each move made by the adversary. When a piece belonging to the adversary is detected, the convolution operation is performed. Varying templates are used by the convolution algorithm to identify all contiguous pieces that are to be chunked. Three separate templates are required to capture the different diagonal movement capabilities of the chess pieces; one for the bishop, queen, and king which can move along the diagonals, one for the pawns which can only move along the forward diagonals, and one for the knight and rook which cannot move along the diagonals.

Each piece that is included in the chunk is further processed by the convolution algorithm. With this recursive use of the convolution algorithm, all of the pieces belonging to a chunk are identified by the convolution algorithm as soon as the first piece is located by the search routine. The convolution algorithm returns a bitmap that indicates the relative positions of all related (i.e. chunkable) pieces. These chunks are then written into a temporary database that is used by the induction algorithm of IAM to detect those chunks that have been repeated by the adversary in more than one game. The purpose of the induction algorithm is to find multiple game usage of specific chunks to verify that a chunk was caused by a definite strategic plan of the adversary and not formed merely by chance.

Induction is used to identify chunks that have occurred in two or more games. The chunks of each game, held in the temporary database, are compared against the chunks of all other games that have been processed, including those chunks already in the adversary model knowledge base. If a chunk is found to be repeated in two or more games, then the chunk is stored in the frame-based adversary model knowledge base. The knowledge base has a slot for the chunk configuration and additional slots for the win/loss result, the color of pieces, the number of times the chunk has been observed, and the time of the game in which the chunk existed (expressed as starting and ending game turns). The additional slots of information are used to improve the quality of knowledge acquired by the acquisition method and to enhance the predictive capabilities of the adversary model. To demonstrate the improvement in knowledge affected by the additional slots, it should be noted that Berliner (1979) has stated that the beginning, middle, and endgame portions of a chess game must be evaluated differently. When a specific chunk occurs repeatedly in games during the beginning and early middle portions of games, then the start and end turn slots are used to prevent the chunk from being used to predict adversary moves during the endgame portion of a chess game.

The induction algorithm is noise sensitive and requires absolute equality between the chunks to be stored in IAM's adversary model knowledge base. Noise intolerance is justifiable for chess and most other adversarial domains, since a small change in the geometry of a chunk will have a major effect on the strategic value of that chunk. An example of this noise intolerance for a three pawn chunk is shown in Figure 2. The first chunk is a strong pawn structure. The next chunk shows two isolated pawns being attacked by an enemy pawn and the last chunk has a doubled pawn which is considered to be much weaker than the first chunk. The difference between each of the chunks in Figure 2 is only a single piece value or piece location.

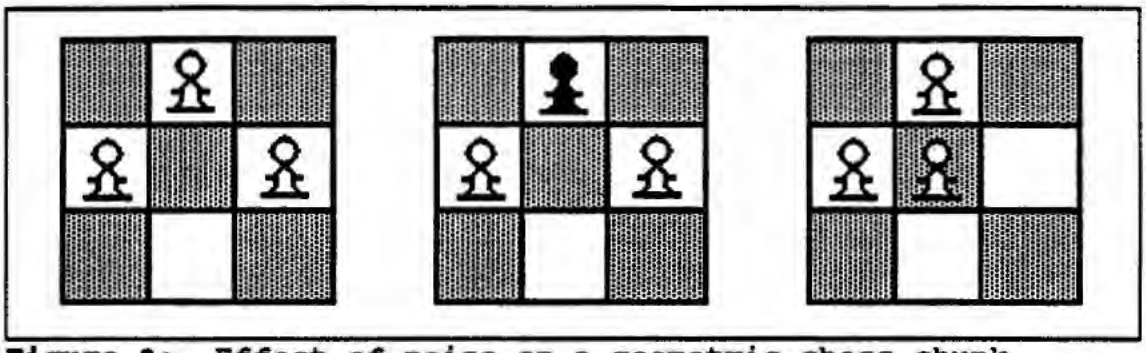


Figure 2: Effect of noise on a geometric chess chunk.

Chunks are considered to be equivalent if they match exactly piece for piece. Additionally, chunks are reflected through a vertical axis when being compared for equality. The vertical axis reflection of chunks is used to capture the left to right invariance of the chess board. A bishop diagonally in front of a pawn is tactically and strategically the same wether it is to the left or right of the pawn. Top to bottom invariance does not exist on the chess board, but this can be modified for domains that have this property, such as Go. A reflection of a chunk through the mid-board horizontal axis is used only when the piece color of each of the chunks to be compared is different. The horizontal reflection acquires similar chunks regardless of the adversary's piece color.

The acquisition of the opening sequences of play demonstrated by an adversary is similar to the acquisition of the chunks geometric piece patterns. The first five opening moves of each game are compared against the opening moves of all other games played by the adversary for identical matches. The color being played by the adversary is of critical importance for the textual move chunks, since the white player is free to follow his own initiative and the black player is responding to the moves of the white player. The same opening move executed as a white player and a black player are considered as two separate moves. A statistical analysis of the opening moves is performed to capture transpositions which produce the same board configuration. The individual moves along with their statistical analysis are stored in another knowledge base in the adversary model.

KNOWLEDGE BASED PREDICTION OF ADVERSARY ACTIONS

The third part of our hypothesis stated in the *Psychological Basis for Adversary Modeling* section forms the premise for utilizing the knowledge acquired by IAM. During future games of the adversary, IAM identifies board positions that are analogous to the geometric chunks that have been identified for the adversary. Analogous board positions exist when a chunk is almost completed on the board. By the hypothesis, the adversary will execute a move that will complete the chunk so that the complexity of the board will be reduced. The knowledge base of opening sequences that simulates a chess masters verbal memory is used to predict which familiar opening patterns the adversary will choose to execute.

Currently, a window size of one move is used for the analogy test. This means that the chunk must be able to have all of the pieces which make up the chunk in their correct relational positions in exactly one move. Board positions that are considered to be

analogous to chunks in the adversary model knowledge base may therefore only have at most two pieces out of position, but are normally required to only have one piece out of position. An extension to the move window size has been added to permit IAM to attempt to eliminate interfering pieces when a castle is the required move to complete the desired chunk.

Because chunks occupy an area of the board, the chunks in the knowledge base frequently contain blank or non-occupied squares within the chunk. These squares serve as wild cards so that the actual square on the current chess board may be blank or contain any other piece. The use of wild cards within chunks simulates the effect of creating larger complex chunks from smaller simpler chunks.

The analogy detection algorithm searches the chess board before each move to be made by the adversary to detect nearly completed chunks. Each chunk in the adversary model knowledge base is compared against the current board configuration. When a nearly complete chunk is detected, the algorithm checks if any legal chess move can complete the chunk. Each chunk which can be completed suggests the appropriate move. The collection of possible moves that enable an adversary to complete a chunk are evaluated by an inference engine to select the best possible alternative.

The inference engine which selects the best possible move to predict the adversary's move choice is heuristically based. The heuristics attempt to apply several psychological principles of cognitive economy and organization, including favoring larger chunks over smaller chunks, not permitting chunks to borrow pieces from a currently existing chunk, and favoring chunks that contain a major, non-pawn, piece. All of the mentioned heuristics attempt to reduce the complexity of the board to the greatest possible extent. If two or more suggested moves have nearly equal probabilities, then the inference engine makes multiple move predictions.

The results of IAM discussed in the next section are all based on simulating the visual and verbal memories of chess masters and their use of these memories in evaluating and solving chess problems. The only domain specific information that is used by the adversary model is the movement capabilities of the pieces and a definition of allowable moves. These domain dependent factors are used to collect geometric chunks of related pieces and primarily to suggest the moves that will complete chunks from the knowledge base. Detailed domain specific information is not required or used to achieve the presented results.

RESULTS

Validation of the knowledge acquisition method we have described is accomplished by using the World Championship games played by M. Botvinnik (Gelo, 1988). The games of the adversary Botvinnik are analyzed by IAM in groups that consisted of all won/lost games from specific tournaments. A game played by Botvinnik in another tournament, Botvinnik versus Petrosian in 1963, is used to simulate a future game played by the adversary. The game used in the test of IAM's predicitive capabilities lasted fifty-seven turns.

The results of IAM's analysis and predictions are shown in Table 1. The text chunks are the patterns found by the opening sequence induction. The first twelve games are from the

1948 Hague-Moscow tournament. The next ten games are from Btovinnik's 1951 World Championship defense against Bronstein. The final fourteen games are from Botvinnik's 1954 defense against Smyslov. From the table, we see that as more chunks are learned by IAM, its performance improves. The significance of the results can be better seen by thinking of the results in terms of American football. From the third line of Table 1, we see that the coach of the football team can accurately predict the exact play of the adversary better than ten percent of the time and can predict the player who will end up with the ball, analogous to the prediction of the correct piece to move, but going to the wrong square, an additional sixteen percent of the total plays in the game. The knowledge contained in the adversary model's knowledge bases gives accurate information about twenty-six percent of the adversary's tactical and strategic decisions with regards to move selection.

Number of Games	Geometric Chunk KB Size	Text Chunk KB Size	Correct Move Prediction	Correct Piece Prediction	Percent of Correct Moves	Percent of Correct Piece
12	28	10	5	8	8.8	22.8
22	53	14	6	7	10.5	22.8
36	66	16	6	9	10.5	26.3

Table 1: Results of IAM prediction capabilities.

Even though the prediction results in Table 1 are impressive, we need to note that the total number of predictions for each of the adversary models in Table 1 is thirty-one, forty-seven, and forty-six respectively. This yields a reliability percentage of accurate prediction to total predictions from a low of 12.7% to a high of 16.1%. The reliability of the predictions is a critical factor to chess evaluation algorithms which would use the adversary model's predictions. A chess evaluation function requires a near one hundred percent reliability to make use of the adversary model's predictions. By using the slot which records the number of times a chunk has been observed, we can produce a base probability of the belief that an adversary will attempt to reproduce a specific chunk. The probabilities are modified by the heuristics of the prediction inference engine to reflect the cognitive principles modeled by the heuristics. The final probability is then used as a likelihood value that a particular chunk will be reproduced by an adversary in future contests. The ratio of accurate predictions to total predictions made by the adversary model is displayed in Table 2, along with the corresponding reliability percentage value.

Increasing the depth of search or correspondingly the search speed of traditional game tree search algorithms augments the tactical performance of the game playing program. Assume that the move generator for a current chess program suggests n moves and searches to a minimum depth of ten ply, or moves by one person. The time order of the search algorithm is n^{10}. The adversary model can predict ten percent of the moves that an adversary will make. If perfect reliability is required, then by using the likelihood values the adversary model can still predict five percent of the moves an adversary will make. With the knowledge available from IAM's adversary model, each time a move is predicted for the

Games	< 25%	25-30%	30-40%	40-50%	> 50%
12	5 / 31 16.1%	4 / 9 44.4%	3 / 6 50 %	3 / 5 60 %	3 / 4 75 %
22	6 / 47 12.7%	3 / 7 42.8%	3 / 6 50 %	3 / 4 75 %	3 / 3 100 %
80	6 / 36 16.6%	3 / 6 50 %	3 / 3 100 %	3 / 3 100 %	3 / 3 100 %

Table 2: Affect of Likelihood on Prediction Ratios

adversary all remaining branches suggested by the move generator can be immedeately pruned from the game tree. This has the effect of reducing the time order of the search algorithm to $(m * p) + n^{(10 * (1 - (p / 2)))}$ where m is a linear constant indicative of the time required by the adversary model to make a prediction and p is the percentage of game moves that are predicted by the adversary model, or five percent for the current knowledge base. The percentage $1 - (p / 2)$ represents the percentage of moves that cannot be predicted by the adversary model. The percentage of adversary predictions is divided by two because the predictions only apply to every other ply of the game tree search. As the reliability and accurate percentages increase and the original depth of the searcb tree increases, the time savings afforded by the adversary model increase proportionaly. With IAM's current performance, one full ply extra can be searched when the original search depth is forty ply. Proportional time savings apply to smaller search depths.

The knowledge about the evaluation criteria and playing style of an adversary grants a significant strategic advantage. In the domain of chess this corresponds to an increase in the depth that is searched in the game tree. Additionally, IAM can predict moves which the move generator of a chess program did not consider. In a seperate test, the commercial programs GNUCHESS and SARGON 4 were given the board position shown in Figure 3, which is prior to the twenty-first move in the tenth win/loss game played by Botvinnik in the Hague-Moscow Championship tournament. Each of the commercial programs was asked to determine the next move to be made by using their standard search algorithms. GNUCHESS chose to move the knight located on the d2 (Queen's second row) square to the e4 square and SARGON 4 chose to move the rook on the f1 square to the c1 square. Using only the geometric chunks induced from the first nine games of the Hague-Moscow tournament, IAM predcited that the knight on d2 should be moved, but that both the b3 and e4 squares were equally likely. The identical likelihood of both predictions is caused by the fact that the same chunk, also depicted in Figure 3, could be produced at both locations by the two separate moves. The adversary Botvinnik selected the Nb3 move. The predictions made by IAM would have alerted the commercial chess programs to the possibility of such a move by the adversary.

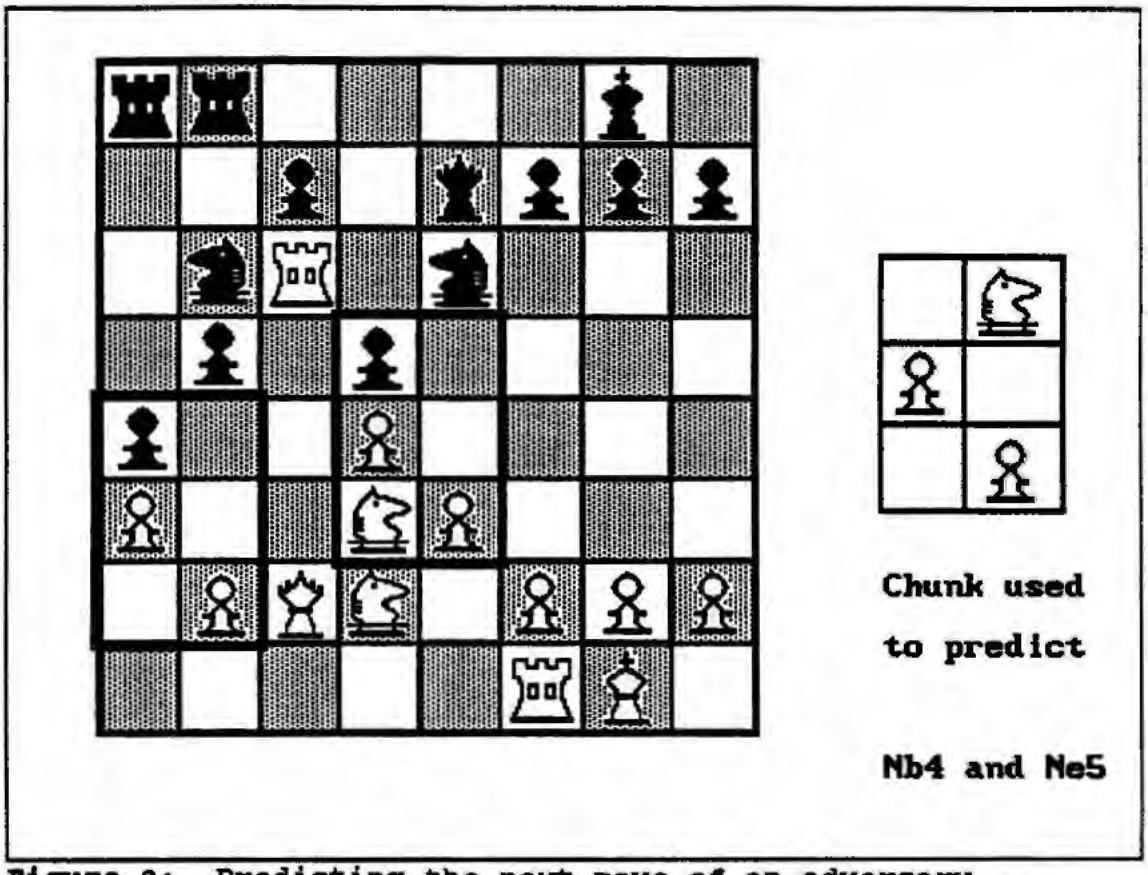


Figure 3: Predicting the next move of an adversary.

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

The performance of intelligent programs is increased whenever additional relevant knowledge is added to the program's knowledge base (Lenat and Feigenbaum, 1987). Berliner and Goetsch (1984) have stated the same principle specifically regarding the adversarial domain of chess. We have presented a method for acquiring and using relevant knowledge that is not domain dependent. The adversary modeling methodology is capable of increasing search speeds and subsequently search depths, which will improve the performance of current chess programs.

Future research will be aimed at increasing the knowledge available through IAM to increase the program's productivity. Both domain specific and domain independent knowledge can be used to augment IAM's capabilities (Walczak, 1990). We have already begun research efforts to implement a minimal amount of domain specific knowledge into the adversary model. By increasing the heuristic rule base of the prediction inference engine to ignore move suggestions made by chunks when a more powerful piece is en prise, able to be captured. With just this small amount of knowledge the total number of predictions made by IAM was reduced by fourteen percent with no loss in the number of accurate predictions. This demonstrates that IAM's reliability can be dramatically increased with the addition of domain specific knowledge.

The knowledge available specifically through the chunk acquisition method can also be increased. Instead of limiting ourselves to geometrically proximal pieces, topological chunks that capture defensive and offensive relationships between pieces that are remotely located on the board can be accomplished. Topological chunks can be constructed by increasing the size of the templates used by the convolution algorithm to cover individual piece movement capabilities across the entire board. For example, to accurately model the movement of a knight we would require a five-by-five template. Another method of incorporating topological chunks that can be investigated is to use semantic networks for knowledge representation instead of the current frame system or modifying the frame system to emulate such a semantic network. The additional chunks acquiring remote relationships between pieces will improve the predictive power of IAM.

Permitting chunks to contain both colors of pieces is also an area for investigation. Combining multiple colors with the distance relationships mentioned above would enable IAM to acquire knowledge about the use of forks, pins, skewers, and possibly revealed checks by an adversary. IAM's performance will continue to increase as new sources of knowledge become available to the adversary model.

REFERENCES

Aliston, W., and Weiskopf, D. (1984). The Complete Baseball Handbook. Allyn and Bacon.

Anderson, J. R. (1980). Cognitive Psychology and Its Implications. Freeman.

Berliner, H. (1979). On Construction of Evaluation Functions for Large Domains. In Proceedings Sixth International Joint Conference on Artificial Intelligence, pp. 53--55.

Berliner, H. (1988). HITECH Becomes First Computer Senior Master. AI Magazine, 9(3), pp. 85--87.

Berliner, H., and Goetsch, G. (1984). A Quantitative Study of Search Methods and the Effect of Constraint Satisfaction. Technical Report CMU-CS-84-147, Carnegie-Mellon University.

Campbell, M. S. (1988). Chunking as an Abstraction Mechanism. PhD thesis, Carnegie-Mellon University.

Carbonell, J. G. (1981). Counterplanning: A Strategy-Based Model of Adversary Planning in Real-World Situations. Artificial Intelligence, 16(3), pp. 295--329.

Chase, W. G., and Simon, H. A. (1973). Perception in Chess. Cognitive Psychology, 4(1), pp. 55-81.

Chase, W. G., and Simon, H. A. (1988). The Mind's Eye in Chess. In Readings in Cognitive Science, pp. 461--494. Morgan Kaufmann.

Davis, P. K. (1988). A new analytic technique for the study of deterence, escalation control, and war termination. Simulation, 50(5), pp. 195--202.

Dunnigan, J. F. (1982). How To Make War. William Morrow.

El-Fattah, Y. M., and Foulard, C. (1978). Learning Systems: Decision, Simulation, and Control. Springer-Verlag.

Gelo, J. H. (1988). Chess World Championships All the Games 1834-1984. McFarland and Company.

Gruber, T. R. (1988). Acquiring strategic knowledge from experts. International Journal of Man-Machine Studies, 29(5), pp. 579--597.

Haywood, O. G. (1954). Military Decision and Game Theory. Journal of the Operations Research Society of America, 2(4), pp. 365--385.

Holding, D. H. (1985). The Psychology of Chess Skill. Lawrence Erlbaum Associates.

Horgan, D. D., Millis, K., and Neimeyer, R. A. (1989). Cognitive Reorganization and the Development of Chess Expertise. International Journal of Personal Construct Psychology, 2, pp. 15-36.

Horowitz, A. (1973). The World Chess Championship A History. MacMillan.

Laird, J., Rosenbloom, P., and Newell, A. (1986). Universal Subgoaling and Chunking. Kluwer Academic.

Langley, P. (1985). Learning to Search: From weak Methods to Domain Specific Heuristics. Cognitive Science, 9(2), pp. 217--260.

Lenat, D. B., and Feigenbaum, E. A. (1987). On the Thresholds of Knowledge. Technical Report AI-126-87, Microelectronic and Computer Technology Corporation (MCC).

Nunn, J., and Griffiths, P. (1987). Secrets of Grandmaster Play. MacMillan.

Riggins, J., and Winter, J. (1984). Gameplan The Language and Strategy of Pro Football. Santa Barbara Press.

Ritter, G. X., Wilson, J. N., and Davidson, J. L. (1988). Image Algebra: An Overview. Technical Report TR-88-05, Computer and Information Sciences Department, University of Florida.

Robbins, J. (1988). America's Red Army. The New York Times Magazine, April, pp. 38--46.

Ryan, P. J. (1988). Some Commandments of Submarine Warfare. The Submarine Review, January, pp. 60--63.

Saariluoma, P. (1984). Coding Problem Spaces in Chess A Psychological Study. Societas Scientiarum Fennica.

Schonberg, H. C. (1973). Grandmasters of Chess. J. B. Lippincott.

Thomas, L. C. (1984). Games, Theory and Applications. Halsted Press.

Von Neumann, J., and Morgenstern, O. (1953). Theory of Games and Economic Behavior. Princeton University Press.

Walczak, S. (1990). Adversary Modeling with Multiple Knowledge Sources. In Proceedings of the Third Florida Artificial Intelligence Research Symposium, pp. 38--41.

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

In what Sense does Knowledge Exist? Alternatives and Consequences for Knowledge Engineering

Thomas Wetter
IBM Germany, Scientific Center
Institute for Knowledge Based Systems
Wilckensstr. 1a, D-6900 Heidelberg, FRG
EARN: WETTER at DHDIBM1

Abstract In a companion paper of Wetter and Woodward [Wette90] there is an outline of a theoretical framework for knowledge acquisition. "Natural" types of occurrence or manifestation of knowledge are addressed there but cannot be discussed in sufficient breadth ond depth. This text tries to give a fine grain differentiation of types of occurrence of knowledge under three essentially different ospects. Examples of the use are then given by characterizing some knowledge engineering approaches in the terms of the introduced differentation, thereby revealing their underlying assumptions.

1.0 Introduction

1.1 Motivation

Knowledge engineering - or engineering knowledge, to bring it more to the point - deals with a "substance" about which very little is known for definite: knowledge. Engineering knowledge has been done in a number of straightforward ways more or less successfully, and in a large number of cases without explicit analysis of what knowledge was assumed to be. It may be argued that once we understand more about the "substance" knowledge, we can do knowledge engineering more consciously and, as a consequence, less erratically.

The following essay tries to illustrate that results from a number of disciplines has brought forth in the last two and a half millennia a variety of ways to look at knowledge. I will try to argue that much more clarity can be gained if we try to describe occurrence of knowledge in the three dimensions ***modality, contents,*** and ***awareness***. As a quick first glance this can be identified with the questions ***how*** we know, ***what*** we know and what we ***believe*** that we know. To argue in favor of this distinction and to provide a large basis of discussion I suggests six possible modalities, six content's, and six awareness's, which can all be traced back to roots in philosophy, cognitive psychology, formal logic, or AI. The aim of this paper is not to come to a final conclusion what knowledge is - and how could that ever be - but to provide a framework for making clearer distinctions about knowledge. This framework being established we can give three-dimensional classifications of existing knowledge engineering methods as to which assumptions about ***their*** types of occurrence of knowledge they make, either explicitly or implicitly. When these assumptions have been identified in the individual case of a knowledge engineering method, a judge-

ment can be made whether the assumptions adequately reflect the to be expected dominating types of occurrence(s) of knowledge in a domain or class of applications.

1.2 Outline of Presentation

First we introduce the suggested three dimensions ***modality, contents, awareness*** of talking about types of occurrence of knowledge (Chap. 2). Principally, any form of modality might occur in combination with any forms of contents and awareness, if the three dimensions were absolutely independent of each other. Chap. 3 indicates some exceptions from this independence of the dimensions and as a side effect may reveal some more details about the dimensions. Chap. 4 illustrates some additional insights that can be gained when making additional assumptions. Chap. 5 applies the dimensions as a method of analyzing a number of knowledge engineering methods by attributing to them their presumable modality, contents, and awareness values. Chap. 6 gives some further perspectives of the disciplines that contribute to making the distinctions. Finally, we discuss, what can so far be covered by the framework and where there are non-covered areas and internal deficiencies.

2.0 Dimensions

2.1 Modality

In what way does knowledge enter a situation or preexist prior to a situation of which an observer would state that knowledge is operational? Is knowledge stable and permanently available or does it need to be triggered? What role does social acceptance of arguments play for how individuals or communities know? These and other questions give rise to the subsequent six distinctions. When so talking about modality, we mean how it really operates or comes into existence in a knower, irrespective of whether we can ever find that out. This may appear like useless hypothetical reasoning. But the application of the introduced distinctions will shed new wavelengths of light on existing knowledge acquisition methods. And, by the way, quantum mechanics could also evolve to become an important theory without principally being able to ever know, of what nature their entities of investigation really are (cf. "Innate part of human machinery -").

2.1.1 World 3 (Popper) without artefact -

World 3[1]

> ***Knowledge is real because it effects the world. Its effect is that the knowledgeable agent acts differently than the non-knowledgeable one.***

This modality goes back to Popper's [Poppe77] claim that something may be real although it cannot be perceived itself, if only it has some effect in one of the (Popper's) worlds. Concerning knowledge, this assumption means that knowledge-

[1] The word under the heading introduces a keyword, which is subsequently used for abbreviation. In some cases the keywords also contribute to the illustration of the concepts.

able individuals can be distinguished from non-knowledgeable ones by differences in behavior, but that remains in the dark what makes the knowledgeable one knowledgeable. This position is not far from Clancey's [Clanc90] reformulation of the knowledge level. One of its aspects is that knowledge is an attribution of an observer agent to the behavior of an (expert) agent, which allows the observer to make sense of the agent's behavior, without any claim of identity between the contents of the attribution and internal states of the agent.

2.1.2 Assemble upon request -

Assemble

> ***An appropriate situation stimulates the expert to assemble the respective body of knowledge.***

This is certainly close to behaviorism (cf [Skinn66]): A stimulus inevitably entails as specific response the recollection, assembly, activation of a body of knowledge. The stimulus is the prerequisite for the knowledge to come into existence. There is, however, one difference to behaviorism. Assembling is no observable event yet. The expert in "assemble upon request-mode" has the choice to exhibit no external reaction, Skinner's probands have no choice. One could argue that sufficiently sensitive instruments could detect action potentials of the brain as the observable manifestation of the assembling. But I would object that the assembly is meant to be a mental state and that there still remains the possibility that numerous different brain states produce the same mental state, i.e. the physical signal representing the assembling cannot be isolated.

Just as a small aside, the example lends itself to illustrate that modality and contents are different dimensions whose degree of dependence or independence will be discussed in some detail in Chap. 3. For now that we are talking about ***modality***, we don't make any claim whether the result of the assembling i.e. the ***contents*** of the resulting mental state is a **theory**, a set of **cases** etc. (cf. "Contents of knowledge").

2.1.3 Mental Heap -

MH

> ***Knowledge is permanent stable part of the expert's mental state.***

Mental models are the classical and presumably most prominent variant of this modality. But while the mental models include the contents related assumption that a map or internal model of the external world is the known entity, we want the modality **MH** to be only characterized by its invariance over standard situations. It needs no stimulus in order to be active, as compared to the assemble upon request modality. Furthermore, the mental heap is selfcontained inasmuch as it does not contain any notion of how to use any of its elements to argue or convince others. It is just permanently there inside the knower.

Over long periods of time and in non-standard situations the mental heap may vary. But this learning issue is not discussed in this text.

2.1.4 Social consensus -

Consensus

> ***Knowledge becomes evident by a discussion, which comes to an agreed upon conclusion. I.e. knowledge is how has to be argued to harmonize two or more views.***

(cf [Winog86]). This may include special forms such as pleadings, explanations in form of monologues, etc. It should be noted that principally social consensus can be achieved without any of the arguments being true or valid in whatsoever objective sense. The only criterion is that arguments are accepted in the involved group. This also implies that in the end the group owns the knowledge in the sense of having an agreement. This does not tell anything about what the group has agreed upon. Considerations by Heisenberg [Heise89] about the social prerequisites for establishment of scientific theories seem to be closely related to this modality. They take e.g. the form of agreed-upon, but not provable conventions about what is an evidence.

Social dominance may characterize phenomena such as guru recommendations for stock exchange etc. This application of the concepts has kind of a strange aspect comparable to a self fulfilling prophecy: A guru recommendations may not be objectively valid when it is said, but can become valid by having been said. This modality is not primarily characterized by internal traits but by inter-individual or social rules about how to argue and convince. Related views are Hegel's dialecticism and Minsky's society of mind (cf [Minsk85]).

2.1.5 Written Documentation -

Documentation

> ***Knowledge has been cast in text, technical documentation, legal code or any other form of mandatory symbolic notation.***

I.e. the user/receiver of the linguistic document has no options to interpret, or, seen from the perspective of the sender, the notation has been given a non-ambiguous form. This has as a consequence that a user of such a knowledge does not have any degree of freedom; he does not act but reacts or executes what is unambiguously prescribed in the document.

2.1.6 Innate part of human machinery -

Innate

> ***If "human" is understood as a species and "innate" as (potentially) collective property of the species, then innate means that potentially all human individuals may arrive at the same knowledge.***

This has a different flavor than **world 3**, because here the being collective is the distinguishing feature, whereas for world 3 the realness of something hidden (which may or may not be collective) is characteristic.

The scope of this modality can be characterized by two extreme examples. The first comes from quantum mechanics, where some experiments can only be understood by assuming a particle nature, others by assuming a wave nature of the entities under investigation. It might be innate that humans can think in terms of particles and in terms of waves. Let us assume the fantasy name "wiple" for the real nature of these entities and further assume that wiple is not intelligible for humans. Then the principle scope of understanding quantum mechanics is innately limited. This does, of course, not mean that all individuals have identical bodies of knowledge, but they share the scope of reachable knowledge.[2] The second example might be used as a counter-argument to innateness, but is used here only for classification. It comes from experiences with schizophrenic patients. They have been observed to develop perceptions and conceptualization of the world, which their psychiatrists are often unable to share. Schizophrenic patients are also human, but their inherited scope of what can be thought can be different from what non-schizophrenic individuals can think.

For those, who may still find this modality strange, the following three protagonists of AI and related philosophical considerations should illustrate that we are in good company.

Fodor's [Fodor88] universal language of thought is one form of saying that all humans share the concepts from which all their thinking can be derived. The same can be said of Schank's [Schan75] 14 (hard-wired) primitives of conceptual dependency, which can be seen as the least redundant formulation of the constituents of a languoge of thought. And finally, Chomsky's claim [Choms65] that there exists a universal mechanism to produce correct sentences from all human languages can be understood as going back to a (collectively) innate machinery which produces the sentences, and whose differentiation towards real languages can be assumed to be different parameters customizing the machinery.

2.2 Contents of knowledge

What is the contents of knowledge? What are known entities, events, details? Do we recall episodes or cases or do we organize in ourselves maps and theories? These questions tacitly introduce not one but two aspects. Asking for *the events* means asking for actual individual entities; asking for how *we organize* means asking for structures. In other words: "Data" and "data structures" [3] are both subsumed under contents. The reason is that "natural" application cannot separate the two. From an analytical standpoint of an observer, who categorizes to his own best what he observes, we could aim at clearly distinguishing. But from the perspective of the 'knower', who just knows and does not reflect his knowledge - this is the one that we try to understand - form and contents cannot be separated.

---

[2] Barbara Becker pointed out to me that these ideas can also be found in Leibniz "Alphabet des Geistes" (Alphabet of Mind)

[3] These technical terms should not be understood in the sense of something hard-wired. But they seem to be the best way to make the desired distinction transparent.

2.2.1 A Theory -

Theory

> ***A collection of abstract concepts (e.g. predicates) and relations between them (e.g. axioms, sub- or superconcepts) satisfying certain quality standards.***

Many of the classical AI approaches can be taken as idealized kinds of theories: production rules, frames, logic theories, conceptual graphs etc. One may ask whether an inference mechanism has to come along with the theory. I take the position to discuss inferences as independent contents type (cf. paragraph **Method**) for two reasons: Formally the same theory may allow quite different inference mechanisms, as exemplified by resolution, sequent, or term reduction type theorem provers for predicate logic. Psychologically, there is a number of open questions and a resulting set of variations whether human inference resembles deduction, natural deduction, induction, etc.

2.2.2 A collection of cases/episodes -

Cases

("case" is used in AI circles, whereas there seems to be a related approach in cognitive psychology dealing with episodic memory.)

> ***The definition has three variants. We may have a collection of cases which are***
>
> - ***encyclopedic complete authentic***
> - ***selective on authentic typical class representatives***

or

- ***prototypical abstractions to represent typical traits of a class***

It should be noted that, despite of the typicality related operations of the second and third alternatives, the basically known objects are the cases themselves, not their attributes, and not at all inferences to argue about similarity or distinctness of cases. One approach in AI, which centers on the cases and uses attributes just for syntactic retrieval originates from Schank's school and was worked out by Kolodner [Kolod83; Kolod84].

2.2.3 Abstract Mechanisms to form Analogies -

Analogies

> ***Methods to transform one situation into an analogous one on the basis of an abstract or symbolic representation.***

This has a typical manifestation in the anglo-saxon case law. Legal expertise consists of analogous transformation of a to be treated case into a precedent case, which has been decided favorably. This reveals the difference to the collection of cases: to retrieve similar cases you just need to know what their attributes ***are***, to transform you have to know what they ***mean***. In the legal case this may mean (and has been attempted by [Rissl87]) to know in what way or direction attribute values may be varied, i.e. sharpened or relaxed in order to still maintain the cases comparable. This is one part of a larger set of possible mechanisms such as neglecting minor devi-

ations, neglecting deviations on irrelevant dimensions, finding one-to-one compensations between dimensions of comparable importance or one-to-many ones when there is one dominating dimension

In contrast to the following, selective perception (which can be understood as being selected by priming by means of prior percepts) does not play a role. All processes described so far operate on a symbolic representation, which is assumed here to be independent of whatsoever perceptive effects.

2.2.4 Perceptive Filters -

Filters

> ***Mechanisms to suppress, transform, amplify etc. elements of a perception, i.e. to memorize not the stimulus itself but a transformation of it.***

Mechanisms of this kind may be effective when interpreting data from mass spectography, seismography etc., i.e. basically analogous signals.

For the moment it does not play a role whether prior percepts/cases, or a theory guide the filtering process. We observe that sub- or pre-symbolic information processing is one characteristic of perceptive filters. This characteristic is shared by the next contents, which finally goes back to Skinner [Skinn66].

2.2.5 Stimulus-Response Patterns -

SR

> ***A stimulus immediately causes a specific response.***

This might characterize tasks of controlling fast continuous processes such as flying a jet. The pairs stimulus - response *are* the knowledge. In contrast to this, the modality **Assemble** is characterized by an enabling condition - the stimulus - whose presence sets free something. This something we call knowledge. Once it has been set free, it can be dealt with irrespective of the stimulus. Skills are not too far from SR-patterns and can for the purpose of this investigation evtl. be taken as identical. There major distinction might be the way how they are acquired by the individual: He is "conditioned" on SR-patterns and actively trains skills. Since this goes into the direction of motivation I cannot go into detail inside the scope of this text.

2.2.6 Problem Solving Method -

Method

> ***Methods to transform problem states into solution states (which requires to recognize problem and solution states as such).***

In a ***normative*** sense this has a tradition back to Aristotle some of which we'll discuss in Chap. 6. ***Empirical*** investigations, however, reveal of a human 'knower', that, if some problem solving method(s) is the essential contents of knowledge then it will considerably differ from well understood formal methods with their completeness and

correctness standards. There are, e.g. results that humans cannot in general produce the consequences from a 1stPL theory (cf. [Brain84]). This does, however, make sense, since methods that characterize human problem solving should have the possibility to make errors - as humans do - which has just been tried to preclude from the formal methods.

2.3 Awareness of the Knower

How does the knowledgeable individual experience or perceive his knowledge? While the issue of modality is how the knowledge objectively is - disregarding whether we can ever finally decide that - and the issue of contents is what are the forms and contents of knowledge, awareness deals with the subjective account of the 'knower' of his knowledge - disregarding whether this subjective view satisfies any standards from outside the 'knower' or whether it is in agreement with the modality or the contents. This can be understood as if in in the sense of Clancey's ([Clanc90]) knowledge level, agent and observer collapse into one person, who attributes rationalisations of his behavior to himself. In this sense the 'knower' "owns" his awareness, and in this very sense it is important for knowledge acquisition: What the 'knower' is aware of can he externalize (to some extent) and offer it actively for knowledge acquisition purposes. It should, however, be noted that most forms of awareness also have a flavor of metacognition: they reflect the reflections of the 'knower' about his knowledge, i.e. they are about the knowledge. This latter aspect will not be pursued systematically.

We distinguish the following types of awareness.

2.3.1 None -

None

> ***The 'knower' does not know either which knowledge he uses nor how he uses it.***

This seemingly implausible assumption may become realistic in a scenario which requires absolutely unshared attention and concentration of the expert in his real working situation. There is no mental capacity left to reflect or self-observe his own behavior. If furthermore the situation is needed to provide access to the knowledge, i.e. there is no retrospective reproduction, then there is indeed no authentic awareness of his knowledge in the 'knower'. This is one of the points that [Dreyf86] make.

2.3.2 To know, when one has the idea -

Referentiality

> ***The 'knower' realizes, how he relates internal knowledge and input from an external source to each other.***

This awareness is in some way extrinsic, as well as the next one. It deals with referentiality, i.e. what entity in the world is referred to by an entity represented internally in the 'knower'. Entities in the world can be perceived as symbolic or subsymbolic, and there is no specification of the type of the external source.

Referentiality can principally not be solved inside a theory, since it will have to go beyond whatsoever theory ([Fodor80] or [Clanc90]). This can be taken as a formal argument why the awareness type **Referentiality** differs from the contents type **Theory**. To put it more application related: Contents = **Theory** is what is principally there and ready for use, whereas awareness = **Referentiality** means that it can be used right now. Without paying too much attention to this expression: Referentiality is kind of a sense of reality of the 'knower'.

Referentiality determines the entry point into an application of knowledge, while the following awareness characterizes the exit from applying knowledge. They have in common the being referred to external entities and hence could be thought of as being one dimension. Again, however, for use of the analysis in a context of knowledge acquisition, the two awareness's are of notably different use.

2.3.3 To know, if something is a solution -

Reflection

Finally evaluate/reflect the application of knowledge.

Whenever an inference has come to an end by means of its intrinsic methods, the result is valid with respect to the internal criteria of the method. The awareness type **Reflection** describes the capability to reflect, whether the formally sound conclusion can also be validly applied to the respective real objects in the domain.

Reflection as a theoretical method has started to be analyzed for use in knowledge based systems as a means to avoid brittleness and to achieve "graceful degradation" instead (cf. [REFLECT]). In this sense, reflection becomes part of the theory. For the reasons outlined above it cannot solve the problems of referring result from inside the theory to entities out in the world which we are talking about here.

2.3.4 To know, what needs to be analyzed -

Evaluation

Know, which (essential) details of a given situation have to undergo application of some knowledge, and, correspondingly, which details can be neglected.

This type of awareness is independent of the fact that the 'knower' may also have knowledge applicable to the non-essential details, i.e. could principally analyze the situation exhaustively. **Evaluation** has a number of variants. In patient care, there may be situations, where therapy can start before a final diagnosis is known. In this case, it has to be known whether it has consequences to further clarify the diagnosis. **Evaluation** means usefulness of further analysis. Judging the balance sheet of a company may reveal certain unsatisfactory but uncritical indicator values and others, which are critical. **Evaluation** now is awareness of sensitivity thresholds above which deviations from expected or required values become critical.

2.3.5 To know, what has to be explained -

Communication

> ***To know what will and what will not be expected in a situation. To explain those details that will not be expected and hence carry a high information content.***

Here the expert is assumed to have an assessment of the normality or non-normality of a detail, relative to a situation and a partner. Non-normality is loosely related to Shannon's ([Shann48; Shann49]) definition of information content. In Shannon's form it makes sense for a purely mechanical receiver, which can't but count and estimate probabilities of events. With a receiver, who is also a knowledgeable agent, Grice's [Grice75] principles are more useful. They incorporate assumptions about the partner's knowledge (and hence: expectations) and suggest to explain, what the partner would not expect on the basis on these assumptions. In any case, the necessity of a detail to be explained does not derive itself from content alone but also from whom to explain.

2.3.6 To know, what is a complete unit -

Formal quality

> ***To treat all details (at least internally) such that in case of externalization the trace would satisfy formal standards.***

At first sight, German criminal law appeared to be an application, where the externalization of a full trace would be needed as an acceptable argument: Legal theory requests that the district attorney proves every detail of a norm as a prerequisite for someone to be punished. (which implements the Roman legal principle ***in dubio pro reo***). I.e. it should be a typical skill of the district attorney to produce formally sound arguments. Practice revealed [Haft87] that - disregarding of the theoretical requirement of formal soundness - default assumptions do indeed enter legal arguments. Nevertheless, the example remains illustrative for the phenomenon.

This awareness may also be understood as metacognition. This would emphasize that it is about the knowledge, whereas the awareness aspect is that the 'knower' has it available.

3.0 Combination of Modality, Contents, and Awareness Parameters

3.1 The Principle of Independent Dimensions

If we claim that modality, contents and awareness are principally independent dimensions to describe different aspects of occurrence of knowledge in the expert, each combination of any three forms should describe a plausible compound assumption. By systematic variation we would in the simplest case arrive at 6 X 6 X 6 occurrence triples

(Modality, Contents, Awareness)

i.e. we would have the possible 196 types of occurrence

(World 3, Theory, None)
(World 3, Theory, Referentiality)
...

There might also occur multiple values on one or more dimensions, such as

((Assemble, Document), Theory, Evaluation)

We will now start by extinguishing a few non-plausible combinations.

3.2 Exceptions: Non-plausible combinations

(Innate, Cases, z)

With an arbitrary awareness z, the origins of the other two are non compatible. **Innate** draws upon human phylogeny, i.e. what all human beings share as potential common knowledge. **Cases** are the different individual random collections of situations encountered. The former denies individuality, while the latter is characterized by individuality.

(MH, Cases, z)

The stationarity of **MH** is in contradiction with the incremental, i.e. time dependent nature of collecting new over new **Cases**.

There may be more implausible combinations, but the examples should be sufficient to illustrate the use of the dimensions in this sense.

4.0 Additional assumptions

For the sake of clarity of argument, we have so far not considered relations between the three dimensions to describe occurrence of knowledge. In other words: In some modality the 'knower' may have some contents of knowledge, which is just a fancy and not related to anything in the world he has to deal with. Or the 'knower' may have some contents or may factually behave according to some modality, but believes that he knows something quite different or feels that his modality is quite different than the real one.

This is not as unrealistic as might seem first. Let us apply once again the argument of Clancey [Clanc90] that a knowledge level description is an observer's attribution to an agent's behavior. Let us furthermore assume that observer and agent are one and the same person. Then the awareness is the attribution of the expert to himself of some knowledge. Let us finally also take Clancey's rejection of Newell's ([Newel82]) identity claim for granted. Then it becomes obvious that agreement of awareness with modality and/or contents could be more the exception than the rule. One may of course argue that on the contrary the identification of observer and agent favors agreement between awareness and contents and/or modality. But this brings us into

more trouble of the kind what can be validly concluded from an observation whose object and subject are identical. So we'd better start as we did, i.e. introduce the three dimensions independently and come now to further conclusion when we start making dependence assumptions.

4.1 Assumption 1: Awareness correctly reflects contents

To make this sure first: Correctness has - as in the case of theorem provers - to be distinguished from completeness: We claim that all that the expert becomes aware of is really on his mind when doing his job. We don't claim that everything that passes his mind becomes aware. And we also don't claim here that, what passes his mind, is true in any objective sense; cf. "Assumption 2: Contents correctly reflects reality".

Furthermore, "correctly reflects" strictly speaking has two meanings: The first and general one is that the 'knower' is aware of the fact that the form of his knowledge contents is **theory** or **cases** etc., cf. "Contents of knowledge". The second one is that he is aware of the individual proposition in the theory, the individual case in his collection etc. The two cannot be strictly separated. The individual proposition as a contents in the stricter sense can only occur in a theory which obviously also has the syntactic flavor of a collection of well formed sentences (cf. "Contents of knowledge").

These clarification being considered, the following combinations have a certain meaning and importance:

(x, Filter, Referentiality)

In other words: the expert becomes correctly aware when and how he can relate sensory input to something he knows. It becomes apparent, with which result the expert transforms a subsymbolic sensory input into a symbolic representation. As a side effect, the expert identifies the instant in time where the pure sensory input starts to make sense for him. In this instant he might be able to name the sense he makes and to identify the material which allows him to make sense, i.e. this identifies an anchoring point for thinking aloud settings.

(x, Analogies, Evaluation)

In other words: the expert knows for what features of a case he solves it makes sense to try and establish an analogy. This identifies the respective feature value as one which is as distant from another case that trying an analogy makes sense. This is not the case for non-deviating dimensions (i.e. identical values) and for widely deviating dimensions. This combination would help to identify the medium range deviations, where forming analogies exhibits its strengths.

(x, Method, Communication)

This reveals methods that the expert feels worth explaining to someone. Reasons for a method to be worth explaining to someone can be

- originality
- necessity due to criticality of a situation
- addendum to a known standard set

Originality might motivate to explain in peer discussions. Because it is the peer who could appreciate the originality. The knowledge engineer will mostly not reach the status of a peer. That is why the expert won't feel motivated to tell the knowledge engineer methods characterized by originality. This may be in favor of a good result of the whole knowledge engineering process: We are maybe not interested in a loose collection of unrelated creative solutions but a sound base equipment of systematic solutions.

Criticality may occur in the (maybe foreseen) need to publicly justify a solution. This will increase the tendency to deal with difficult aspects thoroughly, whereas the standard situation might as well be characterized by a certain negligence concerning partly understood difficult aspects.

Known Standard is not an absolute term but can be seen as relative in a number of ways, some of which can be made constructive use of. One standard might be, what has been described in a book. Another standard might be the set that has already been outlined in prior knowledge elicitation sessions. It is essential that expert and partner (e.g. knowledge engineer) have a common understanding of the standard. This gives an argument, what is the effect of changing the person of the knowledge engineer during the process of knowledge acquisition. Letting the same person do the whole elicitation supports a natural growth of the shared standard of methods. Changing the person stimulates the expert to re-establish the same standard maybe again and again. This may have reasons on the side of the knowledge engineer, but it will have drawbacks about the expert's motivation.

4.2 Assumption 2: Contents correctly reflects reality

The assumption is unnecessary or a tautology in idealistic philosophies: Reality is by being conceived and as being conceived by humans. In any materialistic philosophy assumption 2 is not at all self evident or plausible. One reason is that many domains of application are not completely understood (Whenever natural science is involved, good reason for that can be derived from Popper, cf. [Poppe34].) Even if a domain is completely understood in a community, it may be doubted both, if the expert is mentally capable of individually understanding the domain, and if he is sufficiently motivated to attempt to completely understand the domain. The principal capability is hard to assess in any systematic way (though physics experts admit that operating on the border of abstract nowadays physics concepts puts a hardly bearable mental load on the individual (cf. [Rothe]). The motivation can be readily doubted on the bases of a widespread understanding of mental models as internal models of an external world. The construction, revision, and application of these mental models can be discussed in terms of Piaget's ([Piage67]) accommodation and assimilation. Accommodation, i.e. (re-)structuring of the internal model takes place if and only if observations of the world can by no more means of assimilation be explained by the existing mental model. I.e. incomplete but largely operational mental models persist until at some point in time the expert has seen the whole variety of his domain. This may never be the case in the lifetime of an individual. In any case, it will be hard to argue that a concrete individual has seen the whole variety and can hence satisfy assumption 2.

The outline about assumption 2 indicates that in addition to a clear account of all the variety inside the expert (which is claimed to be assessed by modality, contents, and awareness), a whole theory of knowledge acquisition also needs a clear account

about the validity with respect to the real world. Some of this will, however, be treated in a different text ([Wette90]).

5.0 Characterization of Known Knowledge Acquisition Methods

We now apply the three dimension of occurrence and their values to characterize a number of knowledge engineering methods. In a number of cases we will identify that one or both of the additional assumption are also made explicitly or implicitly by the inventors of the methods.

5.1 Naive Direct Questions

The typical naive optimism of the knowledge engineer asking the expert: "Well, guy, come on, tell me all your rules." is reflected by

(MH, Theory, Formal quality)

with **Assumption 1.**

This should not be taken too seriously, but since it has been the historical starting point of all knowledge engineering, it may still be illustrative.

5.2 Concurrent Thinking Aloud

The basis for the variety of concurrent (and maybe retrospective) thinking aloud methods is

(Assemble, y, Evaluation)

with **Assumption 1.**

Variations are available to provide different triggers in order to achieve different parts of the contents. **Evaluation** instead of **Formal quality** as third parameter is to indicate that complete theories or case solutions cannot be expected but that one may have a patchwork of differently incomplete essentials.

5.3 Observer's Knowledge Level Attributions

This indicates the importance of Clancey's revised analysis of Newell's knowlegde level: If the expert agent has no own account of what knowledge he is applying, a theoretician agent may still attempt to make sense of the expert's behavior in the frame of an assumption that the theoretician makes about contents, cf discussion of epistemological classifications in [Wette90]. I.e. the theoretician supplies for himself a rational bases for explaining the expert's behavior (which - Clancey may forgive me - looks like the observer constructs his mental model of the mechanism underlying the (presumable) regularity observed in the expert behavior.) This can be classified as

(World 3, y, z)

Important corollaries of this perspective are

- The expert agent is only observed, but not involved, i.e. it might be claimed that observation does not modify the observed behavior
- The theoretician agent takes the responsibility for the result. In order to enable him to do so, he must be provided with some rational to argue that ***his*** rationalization of the expert behavior, which prima facie is subjective, can be objectively justified.

  One weak rational of this kind is implicit assumption of the KADS method: A knowledge item is objectively justified, i.e. has received the adequate rationalization or interpretation, when it could be placed in the selected interpretation model. This rational is weak for the following reason: The collection of the KADS interpretation models is itself an attribution of the group of observers in the KADS team to the observations of problem solving behaviors in the KADS experiments. As such, the interpretation models have not been verified by an external independent method, i.e. still remain subjective. Furthermore the selection of the individual interpretation model remains subjective (for reasons see [Wette90]).

5.4 KADS Interpretation Models

(and probably also Chandrasekharan's generic tasks) On the basis of some expert verbalization the knowledge engineer identifies and fills one of the twenty odd methods ("interpretation models" cf. [KADS]), which are assumed to pre-exist before the applications to come (as Newell and Simon [Newel57] once believed that a general problem solving capacity exists before all concrete problems which humans successfully face). In our classification scheme this means

(x, Method, z)
with x = Assemble or MH
with z ≠ None

5.5 Psychometric Methods

The (often implicit) basis or common assumption of all psychometric approaches is : Take a sample of experts, assume that their knowledge is essentially common, homogenous, and representative of the whole population of experts, calculate the "mean" knowledge value (and maybe deviations as a measure of remaining dissent) and take it as the common knowledge (cf. [Boose89; Shaw88]). This can be based on two different assumptions. The knowledge can be common and homogenous as enabled by a common nature of the 'knower's. This is the **Innate** modality from the classification scheme. Or the knowledge can have been harmonized by social exchange and agreement. This is the **Consensus** modality. Hence we have the following two triples

(Innate, y, z)
with z ≠ None

and

(Consensus, y, z)
with z ≠ None

These five notes should only be taken as examples. The general perspective of the suggested classification schemes is to continue analyzing and comparing other existing methods. The scheme attempts to be general. If this claim is approved it can also be applied to methods, to be developed in the future, retrospectively or prospectively, i.e. when beginning to design a new method. For this purpose we subsequently offer some further cues, which disciplines have provided means and eventually can provide further means to better understand the dimensions.

6.0 The Dimensions from the Perspectives of Other Disciplines

The following is to argue that several philosophic traditions, cognitive psychology, and formal logic can help AI in clarifying among some of the introduced variants of existence of knowledge.

6.1 Modality

The theory of knowledge and the theory of science have both contributed to understanding and distinguishing modalities. The theory of cognition has dealt with such questions as whether cognition can grow from observation and distinction or has to be constructed by rational methods. Scholasticism might be one root of this latter approach, which was brought to a climax by the contributors of the Vienna circle (Carnap, Schlick, Wittgenstein etc.). Long before AI, they attempted the formal expressibility of knowledge and an objective approach of deciding about the truthfulness of formal sentences on the basis of reference to the world. Their primary concern was the principle feasibility of the attempt, not the cognitive performance of an individual on such tasks.

Theory of science as another subdiscipline of philosophy can also contribute to understand forms of modality. Authors such as Popper, Bohr, Heisenberg have discussed problems of how to establish or reject an argument or how to distinguish between objectively knowable and observer relative entities whose importance became evident for the first time in quantum mechanics. These are certainly facets of the dialectic modality to come to agreement about a theory.

The contributions of cognitive science to understanding modalities are based on formalizing the inherent structural assumptions of the modalities and to design and conduct experiments to test these assumptions. This may already be apparent by the paragraph about think aloud protocols.

6.2 Contents

The philosophical tradition of normative methods from Aristotle to the Vienna Circle was already mentioned. AI also has a "long" tradition in normative methods, which starts with Newell's and Simon's GPS [Newel57], deals with formal semantics of LISP and PROLOG, and has prominent modern contributions in SOAR and the interpretation models of KADS [Wieli87].

Cognitive science and formal methods from computer science and logic may help each other to reveal, whether the contents more resembles a theory, a set of cases or some of the other variants. The role of the formal sciences is to provide notations of the suspected contents and conclusions that can be drawn (formally) from the contents. The role of cognitive science is to design experiments that offer the chance to decide whether the suspected type of contents is produced, ratified or in any other way (also depending on modality) supported by experience and observation of a human individual.
An example of this might be to assume a theory to be the contents of knowledge, to create a logical theory (in, say, 1st order predicate logic - **1stPL**), to generate the consequences from the theory formally, and to test, whether an individual produces or ratifies the same consequences.
The example is only for illustration and not realistic for a number of reasons. One reason is that there is already experimental evidence from cognitive psychology that 1stPL is too complex to be mentally controllable (multiple negation, sequences of quantifiers, implication, cf. [Brain84]). I.e. to put the assumption "contents equals **theory**" under realistic test, experiments have to use well selected weaker theories than 1stPL. This is just to indicate that in principle cognitive psychology and formal sciences can cooperate to shed light on contents, but that the related investigations have to be thoroughly planned not to lead to wrong conclusions. In the cited example, the theory-assumptions should not be refuted on the base of one (too expressive) language for the theory, but a variety of languages of different shape and complexity should be used, i.e. filled with concrete propositions, and the capability of subjects to predict or ratify theoretically valid implications of the theories should be observed.

6.3 Awareness

Concerning accessibility of observations awareness is the easiest of the dimensions. What the 'knower' becomes aware of is close to be externalized, be it verbally or by some gesture. This should, however, not lead to the conclusion that externalized details are authentic, as has already been outlined when comparing modality and awareness.

7.0 Discussion

In what sense does knowledge exists? The question as a whole is pretty much of the question what intelligence is. The aim of this essay is not to provide the answer. Our concern is knowledge engineering, not the study of intelligence (cf [Clanc90]). However, depending on what assumptions are made about modality, contents, and awareness, knowledge engineering has to be done quite differently. This would de-

velop into a constructive use of a schema, which so far has only been used for analyzing few exemplars of existing methods (cf.Chap. 5). Analysis might also be broadened by not only asking, what the asumptions of the methods of Chap. 5 are, but also whether they are adequate for their purpose. This rises the question of purpose which is worth an investigation of its own.

I did not address systematically the question, whether in what sense knowledge exists is primarily a personal trait, a matter of the socio-cultural environment, determined by the domain about which the knowledge is etc. Some of these issues certainly arise with the modalities **innate** and **social consensus**, but a more systematic treatment might be desirable.

The selection of the three dimensions was a result of thorough investigation and (re-) consideration. It would be surprising if a basically different additional dimension turns up. It would not be that surprising if a new modality, contents, or awareness is detected, or that some two of them are collapsed into one. Especially, I do not see any systematic reason that there must be 6 of each.

This already indicates that an enterprise like this finally aims at a system of categories to describe the phenomenon - knowledge - in sets of terms which are mutually exclusive and exhaustive. I don't claim that this goal has been achieved, but maybe this text stimulates a discussions which brings us closer. This is the more so because the text addresses the question from the perspectives of very different disciplines. It has been argued [Becke90]) that certain insights can only be gained by such an interdisciplinary approach.

Non-plausible combinations in principle indicate that the dimensions are not absolutely independent, as had been desired. Nevertheless, for clarification of the possibilities and for systematic reasons, the introduction of the pure forms first is useful.

This text in a certain way complements a second text of Wetter and Woodward ([Wette90]) which makes an attempt to span a whole theoretical framework for knowledge acquisition and machine learning. The framework is based on a number of cornerstones, one of which is "occurrences" of knowledge. This text here makes a much more fine-grained differentiation of types of occurrence than could be done in the paper spanning the whole theory. On the other hand, this text can't cover implementation and representation issues.

Acknowledgements Yorick Wilks, New Mexico State University, put me on the trace of of our philosophical roots and provided a large amount of references of what to read. This was a major enabling condition for this essay. Franz Schmalhofer, Gabriele Schmidt, Otto Kühn, and Beate Schlenker of the German Research Center for Artificial Intelligence at Kaiserslautern and a number of nameless students at Kaiserslautern University had to suffer from my first attempt and provided some clarifications by their critical questions. In parallel, permanent discussion with Brian Woodward, University of Calgary, helped me keeping the perspective broader then I could have done alone. The second attempt was read by Ralf Nüse and Barbara Heller, IBM Scientific Center, Heidelberg, Barbara Becker, GMD St. Augustin, and Angeliki Polymenakou, London School of Economics, who provided valuable comments. ***Thanks to them all.*** Last but not least, I want to express more than thanks to Angelika, who tolerated that this text was completed late at night of our 15th wedding anniversary.

References

[Becke90] Becker, B. *Interdisziplinarität und KI: Anspruch ohne Wirklichkeit* [4] KI 4 330 - 37 (1990)

[Boose89] Boose, J.H., Bradshaw, J.M., Kitto, C.M., and Shema, D.B.; *From ETS to AQUINAS: Six Years of Knowledge Acquisition Tool Development* Boose, J., Gaines, B., and Ganascia, J.G. (eds.) Proc. European Knowledge Acquisition Workshop EKAW 89, Paris, July 1989, ISBN 2-90367763-8 p. 502 - 515

[Brain84] Braine, M.D.S., Reiser, B.J., Rumain,B. *Some Empirical Justification for a Theory of Natural Propositional Logic* The Psychology of Learning, Academic Press (1984)

[Chomsky65] Chomsky,N. *Aspects of the Theory of Syntax* Cambridge, MA: MIT Press (1965)

[Clanc90] Clancey, W.J. *The Frame of Reference Problem in the Design of Intelligent Machines* in: vanLehn, K. and Newell, A. (eds.); Architectures for Intelligence: The Twenty-Second Carnegie Symposium on Cognition, Hillsdale 1990 (LEA)

[Dreyf86] Dreyfus, H.L. and Dreyfus, S.E. *Mind over Machine* New York 1986 (The Free Press)

[Fodor80] Fodor, J.A. *Methodological Solipsism Considered as a Research Strategy in Cognitive Psychology* Behavioral and Brain Sciences **3**, (1980), 63-73, reprinted in: Haugeland, J. (ed.); Mind Design; Cambridge 1981 (MIT Press)

[Fodor88] Fodor, J. *Why there still has to be a Language of Thought* Partridge, D. and Wilks, Y. (eds.); The Foundations of AI: A Source Book, Cambridge 1988 (MIT Press)

[Grice75] Grice, H.P. *Logic and Conversation* Syntax and Semantics **3**, Speech Acts, Academic Press (1975)

[Haft87] Haft,F., Jones,R.P., and Wetter,Th. *A Natural Language Based Legal Expert System Project for Consultation and Tutoring - The LEX Project* Proc. 1st Intern. Conference on Artificial Intelligence in Law (ICAIL), Boston, May 1987, ACM Baltimore (1987)

[Heise89] Heisenberg, W. *Ordnung der Wirklichkeit* München 1989 (Piper)

[KADS] KADS *Model-Driven Knowledge Aquisition: Interpretation Models* Esprit Project 1098, Memo 87, VF Project Knowledge Aquisition in Formal Domains, Joost Breuker a. o. (1987)

[Kolod83] Kolodner, J.L. *Maintaining Organization in a Dynamic Long-Term Memory* Cognitive Science **7**, (1983)

[Kolod84] Kolodner, J.L. *Knowledge-Based Self-Organizing Memory for Events* A.Elithorn and R.Banerji (eds.) Artificial and Human Intelligence, Elsevier Science Publishers, (1984)

[Minsk85] Minsky, M. *The Society of Mind* New York 1985 (Simon and Schuster)

[Newel57] Newell, A. *Preliminary description of the general problem solving program I (GPS I)* CIP working paper No 7, December (1957)

[Newel82] Newell, A. *The Knowledge Level* Artificial Intelligence **18** (1982) 87-127

---

[4] Interdisciplinarity and AI: Demand without Realization ; (translation by Th. Wetter)

[Poppe34] Popper, K.R. *Logik der Forschung* Wien 1934 (Julius Springer)

[Poppe77] Popper, K.R. und Eccles, J.C. *The Self and Its Brain - An Argument for Interactionism* Berlin 1977 (Springer)

[REFLECT] REFLECT *Reflective Expertise in Knowledge-based Systems* Esprit Basic Research Project P3178, SWI-University of Amsterdam (1988)

[Rissl87] Rissland, E. and Ashley, K.D. *A Case-Based System for Trade Secret Law* Proc. 1st Intern. Conference on Artificial Intelligence in Law (ICAIL), Boston, May 1987, ACM Baltimore (1987)

[Rothe] Rothe, H. Personal communication March 1990

[Schan75] Schank, R.C. *The Primitive Acts of Conceptual Dependency* Theoretical Issues in Natural Language Processing; Cambridge MA ACL (1975)

[Shann48] Shannon, C.E. *A Mathematical Theory of Communication* Bell System Techn. J. **27** 379-423 and 623-656 (1948)

[Shann49] Shannon, C.E. *Communication Theory of Secrecy Systems* Bell System Techn. J. **28** 656-715 (1949)

[Shaw88] Shaw, M.L.G. *Problems of Validation in a Knowledge Acquisition System Using Multiple Experts* Boose, J., Gaines, B., and Linster, M. (eds.); Proc. European Knowledge Acquisition Workshop EKAW88, Bonn, July 88, GMD-Studien 143, St. Augustin 1988

[Skinn66] Skinner, B.F. *An operant analysis of problem solving* Kleinmuntz, B. (ed.), Problem solving: Research, method and theory, New York 1966 (Wiley)

[Wette90] Wetter, Th. and Woodward, B. *Towards a Theoretical Framework for Knowledge Acquisition* to appear in: Proc. AAAI Knowledge Acquisition Workshop, Banff, Nov. 1990

[Wiell87] Wielinga, B.J., Bradeweg, 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

[Winog86] Winograd, T. and Flores, C.F. *Understanding Computers and Cognition: A New Foundation of Design* Norwood 1986 (Ablex)

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

Towards a Theoretical Framework for Knowledge Acquisition

Thomas Wetter
IBM Germany, Scientific Center
Institute for Knowledge Based Systems
Wilckensstr. 1a, D-6900, Heidelberg, FRG
EARN: Wetter at DHDIBM1

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 field of knowledge acquisition is currently characterized by a rapidly increasing number of more or less isolated developments. Researchers and practitioners in the field are contributing to a wealth of techniques and methods, which generate a variety of data from the diverse fields of application. The result is that the field of knowledge acquisition resembles a "patchwork". This paper presents a theoretical framework for knowledge acquisition which provides a common language to describe all attempts in knowledge acquisition, be it isolated methods (e.g. repertory grids) or methodologies, i.e. collections of related methods (e.g.KADS), and to give estimates and make predictions concerning how the methods or methodologies perform. This papers introduces the elements for such a theoretical framework, relates them to the theoretical backgrounds of the involved disciplines by using a four-planes metaphor and attempts to provide an integration of these backgrounds. The framework is then used to assess and describe existing knowledge acquisition methods and methodologies. Future work suggests using the framework for constructing new methods of knowledge acquisition.

1. Introduction

1.1 State of the Art

The field of knowledge acquisition is currently characterized by a rapidly increasing number of more or less isolated developments (for an overview cf. Boose, '89). Researchers and practitioners in the field are contributing to a wealth of techniques and methods, which generate a variety of data from the diverse fields of application. Many of these procedures are adopted from those in other disciplines and are imported into the knowledge acquisition field in order to solve a particular problem, i.e. because they appear promising in a practical sense. Tacitly imported with the individual collections of techniques come their theoretical assumptions and conceptual frameworks from the source disciplines. It cannot be expected that the collection of the frameworks forms a coherent common theory, which could then be suggested as the theory of knowledge acquisition i.e. there is no common theoretical background of knowledge acquisition. There can principally be no common experimental basis either. Since this would have to include standard experiments (i.e. applications to be solved). But once an application has been solved by one (set of) methods all further attempts with methods to be compared would be biased.

The result is that the field of knowledge acquisition resembles a "patchwork" (Shaw and Woodward 1989). Although this characteristic is to be expected due to the age and developmental stage of the field, perhaps enough experience has been gathered to develop a theory for the field of knowledge acquisition. The different methods and techniques used within the field have been drawn from many different research cultures and adapted for use in knowledge acquisition. However, each of these techniques has brought with it a particular set of unspoken assumptions which inform and restrict its use. Until these

assumptions are identified and elaborated, valid comparisons and combinations of techniques are extremely difficult and possibly misleading.

Another point of evidence for the readiness of the field to develop a theoretical framework are the results of the Paris and Banff 1989 workshops: while in Paris there were five papers treating different aspects of such a theory, Banff reflected a dramatic move away from first-generation, stand-alone, fully-automated, locally developed and applied knowledge acquisition tools towards second generation, integrated, theoretically derived knowledge support tools. The Amsterdam 1990 workshop showed the increasing interest towards multiple-tool work-benches based on integrating existing tools in a heterogeneous, pragmatic fashion (Gaines and Linster, 1990) or in a homogeneous, theory-based fashion (Anjewierden et al, 1990). Issues of the integration of knowledge forms naturally accompanied the interest at tool integration (Bradzil and Torgo, 1990; Eggen and Lundteigen, 1990; and Althoff et al, 1990).

1.2 Scope of the Theoretical Framework

It would be the scope of a theoretical framework to provide a common language to describe all attempts in knowledge acquisition, be it isolated methods (e.g. repertory grids) or methodologies, i.e. collections of related methods (e.g.KADS), and to give estimates and make predictions concerning how the methods or methodologies perform. This papers introduces the elements for such a theoretical framework, relates them to the theoretical backgrounds of the involved disciplines and attempts to provide an integration of these backgrounds derived from a variation of the "mystic" "knowledge level" (Newell, 1982, Clancey 1989, 1990; Akkermans, 1990), which is operationalized here as "knowledge perspectives".

The integration is achieved by providing modelling principles and concrete guidelines for the identified fundamental operations. In a next step, a number of well known methods and methodologies are analyzed by means of the suggested theoretical framework, providing some insight into their also well known strengths and weaknesses. In a follow-up paper, we suggest how the framework can be used to construct new, coherent, theoretically founded methods and methodologies.

2. Requirements of a Theoretical Framework for Knowledge Acquisition

2.1 The Four Questions of Knowledge Acquisition

The field of Knowledge Acquisition acknowledges the existence of a 'knower', an individual entity which displays behaviour which is called knowledgeable and which is made available to another knower. To display knowledgeable behaviour, the knower is assumed to utilize a set of information processes of a cognitive nature which are used when the knower interacts with a sub-set of the world. It is important for us then to ask the question:

What happens when an agent operates on a task ?

This question also covers machine learning methods of knowledge acquisition in the sense that a knower becomes active by telling that something is a positive or negative cases or that a system's reaction is' correct'. In order to discuss this question and its possible answers we require a means for identifying and labeling the processes we wish to study. A set of specifically defined concepts are required in order to communicate in a

scientifically valid manner to those in the knowledge acquisition community. We then must ask:

> What are the concepts used to describe and discuss these problem-solving processes and outcomes ?

What makes the field of knowledge acquisition interesting and somewhat unique is the attempts to recreate or model the cognitive processes and/or the outcomes of these processes. This wish to not only understand and study problem-solving processes but to extend the understanding to include representational systems which in some cases perform or 'behave' in similar ways to the knower. We then must ask:

> How are these problem solving processes and outcomes modeled into another medium ?

Finally, the field of knowledge acquisition is required to provide proof that the recreation or model of the processes are good ones; that they are 'correct' re-creations. The last question to be addressed by a theoretical framework for knowledge acquisition is:

> How do we know that our models are correct ?

The first question requires a working understanding of human psychological, especially cognitive, processes. The second question requires the understanding of how we as researchers and practitioners discuss an agent's cognitive architecture, its functioning and outcomes - concepts and characteristics of knowledge. The third question addresses the issues of representation and computation as required by our concepts. The fourth provides, with modelling principles, the basis for analysis, validation and verification.

2.2 Three Basic Requirements of a Theoretical Framework

For working with a theory, we need a common understanding of **language**, a systematic means of producing **conjectures**, and a **frame of reference**.

2.2.1 A language is a set of expressions, which allow descriptions of the phenomena in a field. **The major requirement for a language is sufficient expressiveness.** According to the four questions above, the language must have expressions for the following three cornerstones and their mutual relations:

- occurrences or sources of knowledge (subsequently: occurrences, OC)
- concepts and terms to describe knowledge (subsequently: epistemology, EP)
- formal representations of knowledge (subsequently: computation and representation, CR).

All three corners reflect a two-level structure. On the top level we distinguish *entities* found in the corners from *activities* the knowledge engineer performs concerning the corners. The entities of each corner are further divided into a *generating/carrier 'device'*, an *enabling environment* and *observable manifestations* .

Occurrences

Starting with the *entities*, the generating device is the functional architecture of the expert agent. The enabling environments are real life situations which allow or stimulate him to act such as doing his job, discussing with a peer, etc. Observable manifestations are all externalizations of the expert agent such as utterances, gestures, etc. Another way of

construing this point is to view peer discussion as the enabling environment with the record of the discussion as the observable manifestations.

Activities are closely linked to the enabling environments, since they consist of manipulating the environment and of observing aspects of the environment. The latter is, however, more than recording, since it may include deliberately focussing attention (punctuating the stream of behaviour)

Among the variants of occurrences we will principally distinguish expert knowledgeable behavior from any objective physical manifestation of knowledge containing material. The former addresses all the variants of observing, probing, etc. human beings and will be called psychological occurrence. One of its characteristics is that the observable trace is not machine interpretable. The latter relates to any material that is (or can trivially be made) machine interpretable and can undergo machine learning methods.

We have to take into account a variety of types of psychological occurrences. Some examples are:- experts doing the job, experts theorizing about their domain, experts recalling and externalizing cases or episodes from their experience, human-human dialogues (experts, expert-knowledge engineer, etc.), reading and writing text, diagrams.

There is also a number of physical occurrences, such as:- machine readable case records, machine readable deficient knowledge bases, machine readable operational (but maybe badly organized) knowledge bases. The relation between psychological and physiological occurrences will be analyzed in more detail later. For this investigation, we want them to be seen in common as the available manifestations of knowledge.

Concepts

A *generating device* is a selected class of epistemic concepts, the *enabling environment* is the thought process of exploring the class, and the *observable manifestations* are statements within the scope of the class of epistemic concepts. This may mean that we select, for example, the mental model class of epistemic concepts, our thought process may be guided by introspection or, more realistically in most knowledge acquisition environments, behaviour observation with the aim of making sense of some subject matter, which results in propositions, attached with some truth, evidence, value as the manifestations. *Activities* can be described as operationalized thought processes such as identification operations, induction and abduction steps, etc. Other choices for conceptual class inclusion are episodic memory storage and recall, principles of dialogue behaviour (Grice, 1975), generative models (Clancey, 1990), problem solving methods, labels or descriptions of regularities of a domain, existence of exhaustive sets of cases, or the possibility of providing a representative sets of cases.

Representations

Now the *carrier/generating device* is the formal representation language with an associated abstract inference engine (for practical reasons one can include a piece of hardware on which the language can be automatically interpreted), the *enabling environment* is a concrete knowledge base and a 'case' both written in that language, and *manifestations* are 'traces' of the resulting system. An obvious instance of this logic (eg. first-order predicate logic) with a theorem prover as the carrier device, a logical theory as enabling environment, and a proof (trace) as the observable manifestation. Activities describe the steps of the modelling process to arrive at the final operational system.

We as designers of the theoretical framework can only give examples, i.e. there is no claim that these lists of entities, activities, enabling environments, carrier/generating devices and observable manifestations come close to being exhaustive (as can be claimed either for any of the subsequent lists of examples). But we hope that the lists are illustrative enough to enable the reader to apply our theory to his favourite method or methodology.

2.2.2 **Conjectures,** for a scientific use of the word, take the form of a set of sentences (theorems) about a field, which may be true or false, and which can be applied to phenomena in the field. Conjectures allow us to predict naturally occurring phenomena **The major requirement of a theoretical framework is that it provides the basis for developing meaningful conjectures** (not trivial, but referable to the world) and to compare different theories.

In an engineering theory, theorems allow us to construct objects with desired properties. A scientific and an engineering theory may have a common language: Predicting whether a solid object will collapse under some load vs building a solid object that does not collapse under that load both deals with "solid object", "collapse", "load". But useful theorems will differ in the same way as users of the theory will need different conclusions. One may suspect that scientific use of theory is related to what has been termed diagnosis or classification task, whereas the engineering use is related to configuration tasks or construction tasks, depending on whether the language contains continuous variables.

A central trait of knowledge acquisition is to combine the three corners in a principled way. Hence, conjectures will be formulated later on, which try to combine the terms of the language of occurrences, concepts and formalizations (including their fine structure of carrier, environment, manifestation and activities) in specifically meaningful ways. There will essentially be two major types of theorems: Justifications deal with the entities whereas Modelling Principles (MPs) deal with the activities. For this to be worked out, some other aspects must first be introduced.

2.2.3 A **Frame of Reference** is a set of assumptions which, by convention, are not doubted and serve as guidelines how to observe, whether a theorem is supported or rejected by the world. This includes which expressions of the language to relate to what observables of the field. As being about the theory, the frame of reference cannot be in the theory, i.e. could also be seen as a meta-theory. But for reasons of its functional role, we prefer to speak of a frame of reference. A major requirement for a frame of reference is its social acceptance: the scientific (or other) community has to agree or at least not equivocally reject the way of relating to the phenomenon.

The frame of reference inside our theoretical framework requires that we agree about how to relate manifestations and carriers to each other. This draws upon methods from different fields. Cognitive psychology lends itself for providing criteria for whether some manifestation (utterance, gesture, etc.) is generated by a particular functional architecture. Having a respective frame of reference means knowing, sharing, and uniformly applying such criteria in a scientific or professional community. This is not trivial and primarily not even plausible, because it is somewhat self-referring: if we would all really know what we mean by the particular functional architecture and that utterance A is a manifestation of it, knowledge acquisition would come close to being a solved problem. So we must agree upon a weaker point, namely that our understanding of the terms in the language, and specifically, of manifestations and their carrier devices has sufficient overlap to enable us, as a group, to communicate them.

Agreeing about mutual relations of carriers and manifestations of the concept corner needs philosophical considerations of what is a proposition, a truth value, etc. Knowing

this would enable us as a group to agree that some concrete manifestation is in the realm of a carrier device.

The frame of reference in the CR corner is comparatively trivial. Since we are bound, in any case, to use formal languages and formal operations on them, the set of manifestations supported by the carrier is precisely described as the set of expressions that the formal system can generate (among them the above mentioned proofs).

With the exception of the last point, it should be clear that relating carrier and manifestation has the character of an agreement, which can be withdrawn, varied depending upon cognition and attitude in the scientific or professional community. This is not the typical character of conjectures or theorems, whose claim is to codify insights that are deemed to be general. This does, of course, include the possibility that conjectures are refuted. But, the basis has to be objectively conflicting evidence, whereas a frame of reference can be removed by new agreement in the community about how to observe, interpret, etc.

3. A Theoretical Framework for Knowledge Acquisition

3.1 The Tetrad (Four-Planes) Metaphor

The four questions essentially 'frame' the problem of knowledge acquisition. In order to address the questions and the issues they raise a four-plane metaphor was developed which provides a basis for the language and conjectures of a framework for knowledge acquisition. This framework brings the three corners (occurrences, OC; epistemology, EP; and, computation and representation , CR). into joint partnership for knowledge acquisition. The language of each requires elucidation, points of integration are identified, described and operationalized and a frame of reference established. The concept of a plane was chosen to conceptualize the four questions because planes are useful in providing boundaries, defining multiple relationships, and through the process of defining boundaries, linkages to other planes are established and elucidated.

In the framework presented in this paper, the four questions generate a four-plane configuration (a tetrad). Figure 1. shows in graphic form the four intersecting planes.

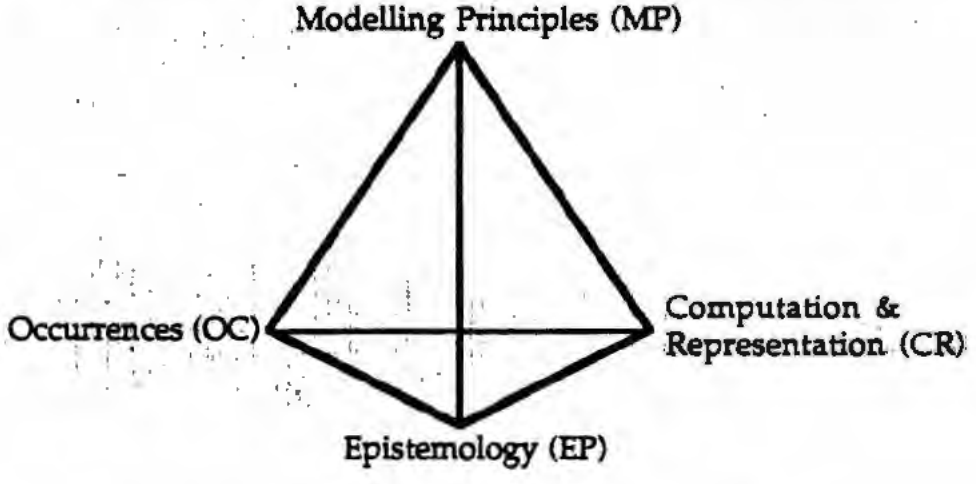


Figure 1: Four Plane Configuration -Tetrad

3.2 Plane One: The Discourse Plane

This plane addresses and essentially defines domain of discourse with respect to the important knowledge concepts used to discuss the agent's mental processes and their outcomes. This plane (Figure 2) stipulates for any proposed theory of knowledge acquisition that those concepts used to speak about knowledge are firmly tied to the assumed functional architecture of the agent. For example, if we talk about 'mental models' then a 'mental model' becomes a legitimate part of the discourse and reflects assumptions about the agent's functional architecture.

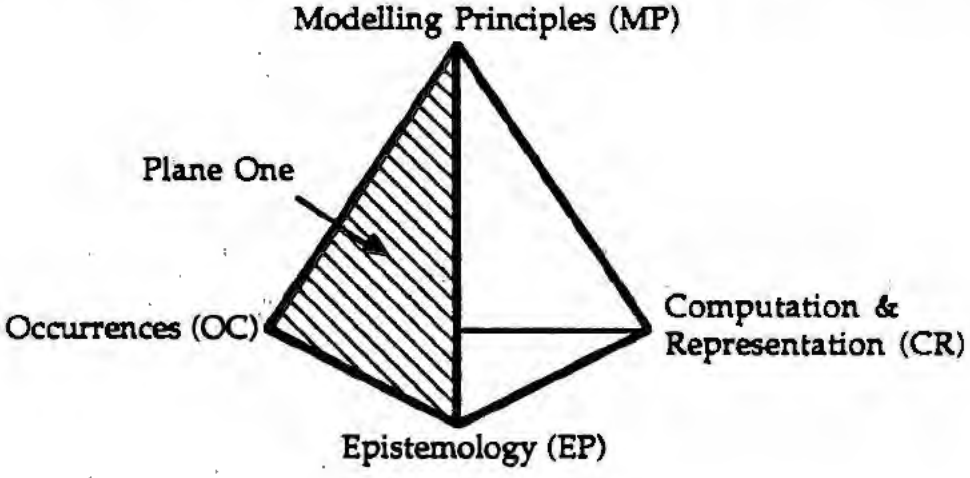


Figure 2: Plane One

The assumptions underlying the use of this concept required elaboration. In order to use the term 'mental model' it may be assumed that there exist an internal (to the agent), unexpressed carrier/generating device acting as a map of the domain. If we talk about this type of representation, then further assumptions follow concerning psychological processes like perception and conceptualization. Having a 'mental model' may assume a memory or storage capacity for information -that the information exists 'apart' from immediate experience. Input to the agent is available for cognitive processing such as matching the input information to the stored information. Perception is relegated to a peripheral, almost passive process (Clancey, 1990).

Further, the 'mental model' becomes something which is used 'like a tool' for the agent to solve problems from the environment. Also, under this concept it might be assumed that the agent needs no specific enabling environment but could spontaneously provide an observable (eg. verbal or graphical form) description of his own 'mental model' which suggests further that an outside 'observer' might also be able to describe this model or parts of it.

On the other hand, if we instead chose the concept of a 'generative model' (Clancey, 1990) rather than a mental model, then we find some clear distinctions. This concept rejects the idea of internal representations (in the agent) of the world as the carrier device. There is no map of the world in the head of the agent. Rather the 'generative model' concept assumes a situation-specific conceptualization or re-conceptualization of the input at hand during problem solving.

Using this concept also necessitates the elaboration and definition of cognitive processes. The existence of an internal memory where information is stored, selected and retrieved is rejected for a set of organizing and arranging processes which are brought to bear on incoming information or which actively selects this incoming information. The role of perception here plays a much more centralized role in the experience of the agent. Perception is viewed as the 'place' where representations are generated (Clancey, 1990). Dynamic representations and their relationships are based on indexical, functional descriptions. Representations do not correspond to objects in the world but are inherently subjective which means that they operate as autopoietic contextually functional operations (Clancey, 1990). This indexical, functional nature, removes the notion of a stored map leaving the point of view focussed directly on the organizing of cognitive processes as the 'site' of 'knowledge'. This last statement is not altogether accurate either. Clancey (1990) points out in his arguments that knowledge-level descriptions are really attributions of an observer rather than 'something' residing 'in' the head of the agent. The scope of this paper does not allow a more detailed analysis of occurrences of knowledge. This has been provided in a companion paper (Wetter, 1990a).

The modelling principles (MP) indicated for this plane reflect the guidelines and procedures used to link discourse concepts and cognitive-level concepts. The modelling principles in this plane suggest that whatever epistemological concepts are chosen, corresponding and/or appropriate cognitive mechanisms be postulated. Because this plane deals with the interaction of the experience or occurrence (OC) and the terms used to conceptualize the occurrence, modelling principles address how we, as observers, select which cognitive processes and outcomes for further consideration and, further, how we define their manifestations. The issue is how, and by what activities, do we map our discourse concepts to those functional attributes of the agent's architecture. Modelling principles suggest, determine or relate cognitive activities to concepts of the domain. Modelling means interpreting expert behaviour in terms of meaningful expressions about the domain. It works differently when we assume mental model or generative model. The result of this modelling is a well-defined discourse domain or Discourse Model which identifies and orders conceptually the important concepts of the theory or methodology.

3.3 Plane Two: The Analogical Plane

This plane is similar to plane one in that concepts are selected to describe and define computational and representational entities rather that cognitive processes. This plane (Figure 3) addresses those principles and activities involved in representation (duplicating, analogizing, etc.) in operational mediums. From the discussion of Plane One, two points of view serve to focus our discussion in this plane.

If, for example, the 'mental model' concept was chosen as part of the Discourse Model (plane one), then it might be assumed that the mental model is in the head of the agent and because it is stored in memory, it is likely somewhat static. Representations in these mental models, then, are considered as objective correspondences to real world entities. Analogical processes could be described as those guidelines and methods which allow relatively static representations in the head of the agent to be maintained in their static nature when represented in another medium. Conjectures about this analogical process would require that the static nature be maintained when re-representing in the new medium. In this case, mental model representations can be incorporated into the new medium if they meet the requirements of the set of analogical primitives. We can say that the knowledge is 'in' the representation. For example, rules, frames or semantic nets are analogous to what is stored 'in' the mental model.

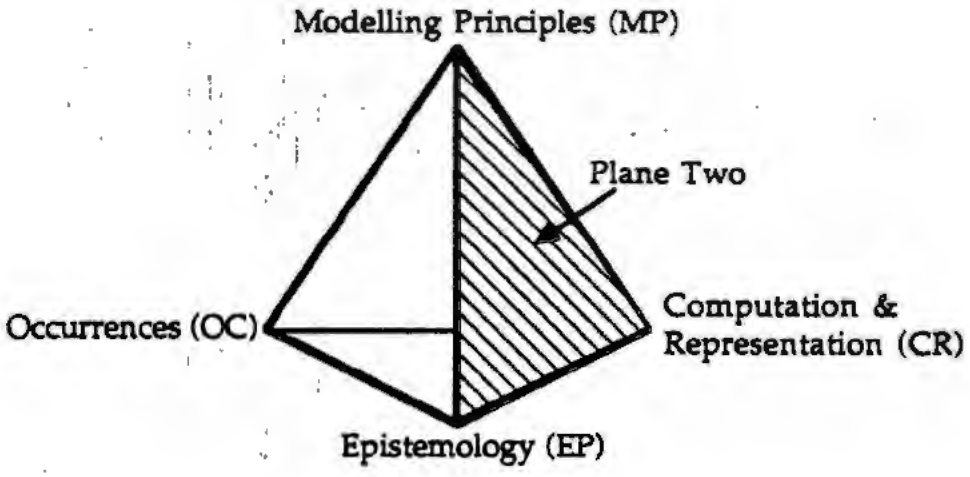


Figure 3: Plane Two

If we accept, rather, the 'generative model' concept (Clancey, 1990), then a set of analogical primitives are still required but they differ in nature (ie.different carrier devices), they differ in what they address and they differ in how we talk about representations. Using the 'generative model' concept, self-organizing cognitive processes generate situation-specific, socially context-sensitive representations, the enabling environment , hence, plays a crucial role. These representations bear no direct correspondences to real world entities but rather, correspond to the subjective meanings required by the agent. Representations are functional in that they operate in a utilitarian manner given contextual demands. Also, they are indexical, suggesting that the representations, which are produced from self-organizing cognitive processes, "call" other processes (they are born from the process and , in turn, trigger them). In this case we are looking at a markedly different set of analogical processes from the 'mental model' concept. Here we want to represent the indexical and functional qualities of representations and we are less concerned with their static form and content. That means self-organizing qualities of groups of processes such as referring, confirming, register-transfer, default, etc. are represented as dynamic and perhaps emergent and are situation-constrained. In this case knowledge could not necessarily reside 'in' the resulting dynamic representations or in the processes generating them. Rather, knowledge concepts are applied or attributed by an observer (Clancey, 1990) to the self-organizing activity: from the situation-specific patterns generated through self-organizing processes.

Here modelling principles involve selecting a formal language or representational framework and deciding what formal language expressions are used for which selected concepts. Determined by what is computationally tractable, assumptions of the target of the modelling become crucial, whereas in plane one assumptions about the source from which to model are crucial. What is similar in either case is the need for a set of analogical processes (an Analogical Model) which are defined by Plane Two definitions and which incorporate the computational tractabilities of the new medium. Operational requirements and constraints of the mediums require identification and description which are embodied within a set of analogical primitives. This set is strongly influenced by the concepts chosen and defined in the Discourse Model and by the computational tractability of the mediums. The concepts outlined in the Discourse Model are shared by planes one and two and act as the functional point for the creation of analogical links between cognitive processes and operational representations.

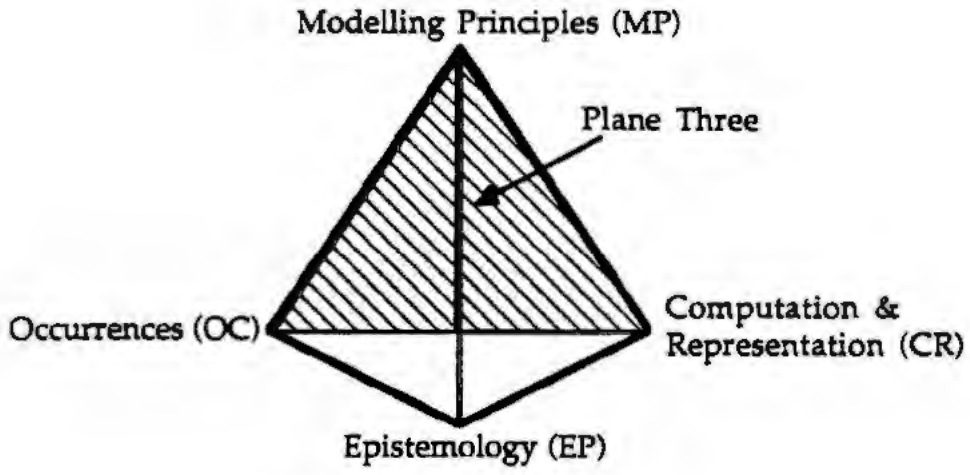


Figure 4: Plane Three

The models generated from this plane are operational in nature (Operational Models). They guide tool development, focus and operationalize the models of the other two planes. Modelling principles in this plane address how the patterns identified and labeled are to be translated from one medium to another. These principles are similar to those in the activity of design and in the activity of engineering.

Once the Discourse and Analogical models have been developed , the basis for Operational models is determined. Without these models a 'compiled' approach is taken to developing knowledge acquisition tools. In a compiled modelling step the 'cognitive into conceptual' and the 'conceptual into formal' steps are combined into a single step. It is possible that many existing KA tools and techniques are manifestations of such compiled modelling steps, which might be better understood, when Discourse and Analogical models are developed prior to the development of Operational models.

3.5 Plane Four: The Coherence Plane

All planes discussed so far deal with the activities required to link their respective base plane corners. The role of plane four is two-fold: it implicitly serves as a basis for all the modelling principles and it provides a frame in which to justify the result of modelling (Figure 5). This plane frames the dimensions on which we conceive, understand and construct knowledge acquisition methods. The epistemology point (EP) requires that we select a sub-set of terms (representing concepts) which we deem to be in accordance with properties of the agent's functional architecture that generates the observable manifestations. The occurrence point (OC) reflects a sub-set of all cognitive processes available to the agent which are used to solve a particular problem under study. The computation and representation corner (CR) represents a sub-set of what is computationally tractable. This plane provides the basis for discussing the degree of coherence among the knowledge-level/cognitive-level elements in the Discourse Model, the knowledge-level/ computational-level concepts in the Analogical Model and the computational-level/cognitive-level concepts in the Operational Model. Coherence reflects the basis for a validation and verification framework for knowledge acquisition tools and for the content models generated using the tools.

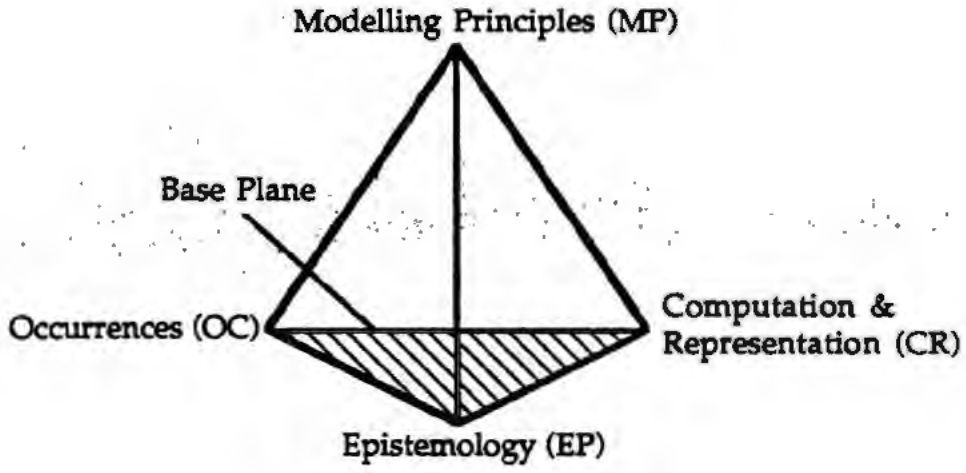


Figure 5: Base Plane

3.6 A Comment on the 'Knowledge-Level'

Using the four-plane model as a basis for the theoretical framework, the issue of knowledge-level can be re-addressed. The term 'level' appears to misname the phenomenon we are attempting to study. The term misdirects our attention. "Level" implies a hierarchy so when one reaches the Knowledge Level, then one is able to comment on what is observed. In Clancey's case (1990), the Knowledge Level refers to the constructions (concepts/terms) used by an observer to order or to refer to a pattern of behaviour in the artifact under study. Newell (1982) views the Knowledge Level as concepts used to refer to structures in the artifact which support knowledge . 'Level' also implies that one has reached the 'most advantageous' or 'best' point from which to discuss and/or observe knowledge structures.

Perhaps a less inhibiting term is "perspective" such that there exist a number of perspectives; knowledge perspectives. This term implies a 'point of view', a 'vantage or observational point', a 'stance'. We can discuss, then, an observer's perspective and the requirements and characteristics of this perspective. We can discuss the 'subjective' stance of an individual who knows he knows and we can discuss knowledge from the perspective of the external representation.

The observer's perspective requires a set of constructs/methods for labeling and discussing patterns of behaviour of intelligent artifacts. The second perspective arises from the individual and is supported by writing in phenomenology and constructivist philosophy. The third perspective requires elaboration. The knowledge perspective suggested by the external representation contains 'aspects' of knowledge. This almost sounds absurd and contrary to much discussion (eg. Hirst and Regoczie, 1989; Regoczie and Hirst, 1989). However, if we include in this perspective the user of the artifact (text), then we can describe certain structures contained in the artifact which 'trigger' knowledge or cognitive processes in the observer/user. The act of comprehension (Graesser and Clark, 1985) is triggered by linguistic structures or diagrammatic ones. Clancey's point that all knowledge level discussions require an observer is maintained but the observer's stance is qualitatively different depending on which of the three perspectives chosen.

3.4 Plane Three: The Operational Plane

This plane directly addresses the issues of translating identified and defined cognitive and 'learning' manifestations into computational and representational mediums. This plane (Figure 4) addresses the 'method' procedures of knowledge acquisition. This plane describes whether the cognitive processes and physical occurrences which were chosen are computationally tractable and meaningful and defines the techniques and methodologies which are required for knowledge acquisition. The basis for the techniques are defined from the other two planes while this plane addresses those specific activities necessary to accomplish a re-representation or re-creation in another medium.

The decisions made on the other planes influence the types of techniques and how they are developed or imported from other disciplines. For example, if the 'mental model' concept is used, then it is assumed that the mental model is in the head of the agent, the mental model is used to solve problems, the mental model can be identified and described by the agent, and representations 'in' the mental model can be re-represented or constructed in another medium which maintain the static structure of the 'knowledge'. These beliefs let the tool developer or tool user talk about knowledge acquisition. Tools developed using these assumptions centre on the 'extraction' concept of knowledge acquisition.

If the concept of 'generative model' is chosen, then it is assumed that knowledge is something attributed by an observer (knowledge engineer) based on patterns of behaviour of the agent. Knowledge descriptions are EP corner manifestations of the observer's labelling of agent behaviour manifestations, the agent's 'expertise' is displayed in interaction with the domain so that emergent properties in behaviour are possible and likely, and autopoietic processes are indicated in problem-solving processes (Clancey, 1990). The agent may not be able to articulate and describe these self-organizing processes but rather indicate the cues (indexical and functional associations may or may not be apparent only the results of these).

Tools developed using these assumptions focus on the modelling the dynamic aspects of knowledge. Patterns of inference processes and behaviours, conditions of the environment of, and observations of the occurrences of knowledge in a natural setting are considered. These assumptions also indicate a focus on modelling the knowledge engineer's labelling and attribution processes when 'punctuating' the stream of behaviour. Some aspects of protocol analysis (e.g. Johnson, 1986) reflect 'generative-like' assumptions about knowledge. Compton and Jansen (1990) argue for the development of knowledge acquisition tools which support their epistemological assumptions that knowledge is only knowledge when it is used (acquired) in context and that knowledge acquisition tools must retain this context.

4. Assessing Knowledge Acquisition Methods and Methodologies

The theoretic framework developed above provide a basis for assessing knowledge acquisition methods and methodologies. We make the assumption that the language of OC, EP, CR, and MP, together with their fine structures are sufficient to describe approaches to knowledge acquisition (this assumption is further discussed in 6.1). We can assess methods through the use of all the elements of the theoretic framework: the language of each corner, the conjectures relating elements of the corners, and the frame of reference assumed to be shared by the community. These approaches of assessment are outlined in the following sequence:

Assessment by Coherence - Justification Conjectures
Assessing the Corners - How is the language used?
Assessment by Linking Activities - Modelling Principles
Assessment by Plausible Assumptions - The Frame of Reference

In the next section the theoretic framework developed above is used to analyze examples of existing knowledge acquisition methods and methodologies. The following analyses do not represent a prescribed process but rather a description and analysis of existing knowledge acquisition theories as understood and portrayed by their methods and tools. The areas of description and assessment correspond to the three requirements outlined above: the language, the system of conjectures and the frame of reference. We as designers of the theoretic framework can only give examples, i.e. there is no claim that these lists even come close to being exhaustive. The lists of criteria are illustrative to enable the reader to apply our framework to his/her favourite method or methodology.

4.1 Assessment by Coherence - Justification Conjectures

Depending on the corners we have to assess and the ways they can be assessed, we distinguish the following four classes of justification conjectures (which for convenience of presentation are written as questions).

Face Justification is addressed by the EP-CR line. This asks the question 'Does the result of CR "look like" the concept chosen in EP ?'. In other words, does the representation of the objective artifact reasonably resemble what is meant by the EP terms used to describe it ?. Would others agree ?

Construct (Concept) Justification is addressed by the EP-OC line which asks if the term or concept selected is supported by psychological evidence and is the term used consistently in the KA context as it is used in the psychological research ? Does the term describe a known phenomenon?

Concurrent Justification is addressed by the OC-CR link and requires that the CR product is a reasonable artifact of the original PS phenomenon. This is really validity which comes from the main assumptions of Cognitive Science. Does the CR artifact demonstrate a reasonable explanation of the PS experience? Does the artifact explain or 'behave like' the cognitive phenomenon and does it replicate the phenomenon and does it support the results with human subjects?

Predictive Justification is addressed also by the OC-CR link in that with new or different (out of range) input to the CR artifact, will the results predicate or suggest alterations to the OC phenomenon.

It should be noted about the character of these justifications that the justification between OC and CR is based on observable manifestations, whereas the justifications combining EP with the two other base-plane corners are based on the carrier/generating device. In other words, justification cannot be done without having the frames of reference involved. This really focuses the endeavour. Since the frame of reference of the CR corner is trivially determined by formal properties, the OC frame of reference has already been identified as the crucial one which would require more knowledge about knowledge than we can ever have.

Once coherence is established, then it can act as the basis for establishing Validation Models for the CR artefact in the real world. It is important that any knowledge acquisition method is accompanied by a model or set of procedures for its own validation in the real world of users. This is often lacking. Based on a coherent development strategy, the generation of validation models becomes ordered and principled. Coherent development anchors the artefact or method for validation and verification. For example, if there exist two distinct occurrences (OC1 and OC2) for the same term (EP) and representation formalism (CR) then OC1 can be used to validate OC2.

4.2. Assessing the Corners - How is the language used ?

This assessment is described in a great deal of detail since it provides the indispensable base for the next two assessments.

4.2.1 Empirical Observations for Psychological/Cognitive Classifications (OC)

The agent is capable of demonstrating many forms of interaction with the target system (task or problem) and it is assumed that each form of interaction demands a variety of cognitive processes in order to successfully address the task. Shaw, Woodward, and Gaines (1990) have identified five main categories of cognitive activities used by an agent while addressing task demands. These five categories were developed to identify a variety of cognitive processes that correspond with classes of knowledge elicitation and acquisition tools. These categories are being used here as examples only. They do not cover the whole range of psychological processes possible, only those from an elicitation point of view. It is not suggested here that all knowledge acquisition tools/methods must meet each of the requirements, only that the use of the tetrad framework can be operationalized in one way by using these categories.

a) Interaction: the set of cognitive processes used while the agent is actively involved in addressing the task demands. These processes may involve the performance of physical as well as cognitive skills. Interaction may be captured by behaviour observation. This may take two forms: the agent acts or interacts (e.g. behaviour may be observed and recorded or not).

b) Introspection: the set of cognitive processes in operation when the agent reflects on the skills and processes involved in addressing a task and/or the awareness of self as the task is being addressed. Certain elicitation procedures (eg. repertory grids) prompt these same processes. To some extent the agent introspects and evaluates the knowledge (e.g. recalling cases or episodes from experience and/or theorizing about the domain).

c) Communication: the set of cognitive processes involved while the agent communicates aspects of the task or aspects of traces of his/her own cognitive processes to another individual during or after the task is addressed. Discourse-based methods have been used with these types of processes (e.g. human-human dialogues).

d) Formalization: the set of cognitive skills involved in formalizing concepts and relationships within a body of information which results in a well-formed, structured, internally coherent and consistent set of statements. Logic-based languages reflect these types of cognitive skills (e.g. when the expert is asked for uncertainty weights or rules).

e) Explication: the set of cognitive processes involved in the detailed analysis and logical extension of a body of formalized concepts and relationships. This includes the processes involved in developing implications based on formalized relationships (e.g. expert is asked about a new case or a situation on the fringe of the domain).

Most of these activities may provide data for both psychological and machine learning assessments. Explication can, for example, lead to a psychological occurrence of externalizing a mental model or as a desired proof for a learning-by-failure-to-prove algorithm.

4.2.2 Computational/Representational Classifications (CR)

The forms and formalisms required for representation and computation can be classified into four main types (Shaw, Woodward, Gaines; 1990). Each type is differentiated based on certain characteristics of the representation. Most of the representational schema outlined are well known in AI.

a) Behaviour Data-Base: observations of the agent interacting with the target system (addressing the task demands) generates a database of behavioural data. The data may take the form of cases or examples and are stored together as a composite database representing close to the 'raw' data.

b) Informal Knowledge Base: multiple-form representations of a linguistic and/or diagrammatic nature are co-ordinated to constitute tentative or partial 'pictures' of the agent addressing the task. This type of knowledge base demonstrates conceptual links with a minimum of formalization. They are usually very rich and complex.

c) Formal Knowledge Base: usually single-form representations with a high degree of structure and denotational in nature. They are usually very specific in content and well-formed structurally.

d) Computational Knowledge Base: a formal knowledge base with computational formalisms and procedural in nature. This type of representation is capable of generating 'behaviour' based on a variety of algorithmic procedures and declarative structures.

4.2.3 Methods for Epistemological Classifications (EP)

In order to discuss the characteristics of both the cognitive processes and the resulting representations a set of concepts are needed which are agreed upon and which are deemed to refer to those selected characteristics. This is an important step in the development of any method of KA because it directly relates to the selection and/or development of analysis procedures used with the representations to generate 'meaning' from those representations and computations.

a) System Identification: aspects and characteristics (parameters and values) of the agent's interaction with the target system are identified, labeled and defined. No attempt is made to impute 'meaning' other than through naming and location.

b) Empirical Abduction: patterns are identified and labeled based on behavioural interaction. The patterns are the source of 'meaning' because to identify them suggests an understanding of the relationships underlying the behaviour.

c) Conceptual Analysis: the focus of attention is on conceptualizing the structures and processes which produce the patterns of behaviour. The 'meaning' is assumed to be created through an elucidation of the mechanisms which produce the patterns.

d) Knowledge Replication: these processes focus on the 'meaning' created by a replication through imitation or emulation of the conceived processes underlying patterns of behaviour of the agent.

4.3. Assessment by Linking Activities - Modelling Principles

This assessment is based on whether or not the models or methods under consideration address the modelling principles involved in each of the three 'modelling' planes of the tetrad framework. In other words, have the developers outlined their modelling principles for:

1. selecting their terms and linking these terms with occurrences of knowledge (Discourse Models)
2. selecting their terms and linking them with representational/computational forms (Analogical Models)
3. linking the selected occurrences of knowledge to aspects of the artefact (Operational Models) ?

We cannot give much detail of this here in the scope of this text; however, the reader can refer to the preceding section (4.2) where the activities which need to be linked have already been described on their own (i.e. 4.2 taken as a reference can be used to address 4.3). Let us consider one differentiation between activities related to OC as compared to CR and EP. The activities a through d of both CR and EP can be viewed as stepwise creative refinements (i.e. they have an underlying common time sequence). They can be linked on the basis of their sequence of results. OC, in contrast, is described by an unordered collection of interactions and by what can be observed from them. Hence, there is no one-to-one correspondence between any of the OC and any of the other activities. A reason for linking has to be provided individually by the knowledge engineer anew for every new application.

An instance of well-founded linking is the work on repertory grids (Gaines and Shaw, 1990). The Discourse Model is developed from the selection of one epistemic concept- "distinction". The principled selection of this term and resulting terms is clearly and substantially supported by the extension of Kelly's (1955) personal construct psychology. The epistemic term 'distinction' (and subsequent 'systems of distinctions') is clearly linked and clearly coherent with the psychological occurrences outlined in Kelly's theory. Furthermore, the linking of the OC and EP corners of framework is principled, orderly and logically developed.

With the Discourse Model firmly developed, Gaines and Shaw go on to develop a set of knowledge representation formalisms based on the term 'distinction' and its derivations. Based on (derived from) principles underlying a system of distinctions, a system of representational formalism was developed (concept, individual, role, property, attribute, relation, etc.). This Analogical Model is, once again, developed logically and has been made explicit.

Finally, in a companion paper, Gaines (1990) extends the Discourse and Analogical Models to define an Operational Model. He outlines the use and integration of and integration of a number of knowledge acquisition tools which are based on the underlying systems of distinctions. The methods outlined for eliciting, developing and connecting distinctions is presented.

When all three models have been developed in a principled fashion, 'coherence' is firmly established. This is very evident in the work by Gaines and Shaw (1990).

4.4 Assessment by Plausible Assumption - Frame of Reference

In this form of assessment, a major requirement of a theory of knowledge acquisition is that its terms and their definitions experience a high degree of social acceptance. The scientific (or other) community has to agree or at least not equivocally reject the way of relating to and testing against the world. This includes which expressions of the theory relate to what observables of the field. Our understanding of the terms in the language of the theory should have sufficient overlap to enable knowledge acquisition researchers, as a group, to apply, then to functionally compare, phenomena in the world. Illustrations of this assessment area are given in the next section which gives examples of using the theoretical framework with three knowledge acquisition approaches.

5. Three Examples of Classification

KADS

1. Assessing the Corners

1.1 Empirical Observations of Psychological/Cognitive Classifications

This is not addressed in KADS. The developers have abandoned the aim of producing an image of what the agent does but rely on a re-creation that has the required functionality but essentially is the knowledge engineer's conceptualization of the agent in interaction with the target.

1.2 Computational/Representational Classifications

1.2.1 Behaviour Data-Base: This is supported by KADS' Power Tool workbench and its successor, the Shelley workbench.

1.2.2 Informal Knowledge Base: In the early phases of KADS, the conceptual model was an informal knowledge base. This had a number of disadvantages. hence, there are now two logic-oriented formal languages for the conceptual model (Akkerman et al, 1990; Wetter, 1990b)

1.2.3 Formal Knowledge Base: This is not present in KADS. The formal notation (see 1.2.4) entails a procedural semantics- it contains explicit control expressions.

1.2.4. Computational Knowledge Base: Formerly, the KADS knowledge base only became executable after the design phase. With Wielinga et al's (1989) and Wetter's (1990) contributions, the conceptual model can be made executable without being a rapid prototyping approach but representing a true, but probably very slow, image of the future artifact.

1.3 Methods of Epistemological Classifications

1.3.1 System Identification: The KADS paradigm of the conceptual model with its four layers is the prescribed system of identification for a knowledge engineer in KADS. This system requires a collection of preformed interpretation models (i.e. preformed prototypical methods to solve all problems in a particular class). Complementary to this, there are application specific details against which are applied the inference steps from the interpretation model.

1.3.2 Empirical Abduction: Not used.

1.3.3 Conceptual Analysis: The KADS method uses the 'slots' from the best fitting interpretation model (out of a library of some 25 interpretation models). Observations or 'utterances' are related to the 'slots' in the selected model. In this manner, meaning is supplied for the utterances.

1.3.4 Knowledge Replication: A formal, operable, replication comes after the conceptual work is completed and is not directly covered by the KADS methodology.

2. Assessment by Linking Activities

2.1 Plane One (EP to OC): There cannot be any because KADS does not specify any cognitive processes. This leads to the specific deficit that there is no feedback from intermediate formalisations to the best fitting elicitation method. KADS gives a hint as to 'what' has to be elicited but one can only know 'how' to elicit what is required if a cognitive classification is available. This classification is lacking in KADS.

2.2 Plane Two (EP to CR): The procedural semantics of the conceptual model must be identical to the knowledge engineer's understanding of the interpretation model. The functional artifact must do what the knowledge engineer believes it does with his selected interpretation model.

2.3 Plane Three (OC to CR): There are none. This is in accordance with no expert agent having an occurrence connected directly with a formal representation. KADS relies heavily on the knowledge engineer to make the 'right' attributions and to understand how the computer language artifact interprets these attributions. On the other hand, KADS supports very efficiently the knowledge engineer in his/her job.

3. Assessment of Coherence

3.1 Coherence (OC to CR to EP): Due to the neglect of the occurrence analysis, KADS cannot be coherent as defined by our framework. It might be made more coherent by means of reasonable assumptions about what mechanisms produce the occurrences as they are used by the KADS method.

4 Assessment by Plausible Assumptions - The Frame of Reference

4.1 Frame of Reference: The knowledge engineer believes that there is an interpretation model which 'fits' and that his/her observations 'fit' into that model.

PROTEGE

1. Assessing the Corners

1.1 Empirical Observations for Psychological/Cognitive Classifications

1.1.1 Interaction: This does not occur.

1.1.2 Introspection: The expert introspects about the forthcoming skeletal plan

1.1.3 Communication: The expert communicates (interprets) his/her plan with the knowledge engineer

1.1.4 Formalization: The knowledge engineer formalizes the plan.

1.1.5. Explication: Does not occur because the skeletal plan is limited to very specific topics and attributes of the problem.

1.2 Computational/Representational Classifications

1.2.1 Behaviour Data-Base: Not needed because the target representation is so definite that no formal pre-processing is required.

1.2.2 Informal Knowledge Base: Not needed because the target representation is so definite that no formal pre-processing is required. If PROTEGE someday aims at solving other than skeletal plan refinement tasks, a need might occur to support prestructuring.

1.2.3 Formal Knowledge Base: Forms are developed by the knowledge engineer to reflect the requirements of the user.

1.2.4. Computational Knowledge Base: The semantics of the skeletal plan are so clear that the code can be generated automatically.

1.3 Methods for Epistemological Classifications

1.3.1 System Identification: The only system to be identified is skeletal plan refinement.

1.3.2 Empirical Abduction: This plays a minor role. The expert provides a few cases about the problem area.

1.3.3 Conceptual Analysis: The two major concept classes identified are the structural and incidental details of the plan.

1.3.4 Knowledge Replication: This is the newly created skeletal plan.

2. Assessment by Linking Activities

2.1 Plane One (EP to OC): The result of introspection and communication must be understandable as a skeletal plan.

2.2 Plane Two (EP to CR): Conceptual analysis (1.3.3) results must match formal knowledge base structures (1.2.3) and the meaning of the skeletal plan (1.3.4) must match the machine interpretation (1.2.4) of the formal knowledge base (1.2.3).

2.3 Plane Three (OC to CR): There must be adequate data structures in the Formal Knowledge Base to accommodate the information which comes from introspection and communication. The expert and knowledge engineer need to agree on and verify these information points as meaning the same thing. Also, the automatically generated code must correctly reflect these meanings.

3. Assessment by Coherence

3.1 Coherence (OC to CR to EP): The plan items provided by the expert must match the knowledge engineer's concepts of skeletal plans and have one-to one data structures in the PROTEGE plan generator.

4. Assessment by Plausible Assumption

4.1 Frame of Reference: In contrast to KADS, this frame of reference needs to be shared between the expert and knowledge engineer. They must both understand the common characteristics of skeletal plans and the instances of a specific plan.

REPERTORY GRIDS (Aquinas, KSSO)

1. Assessing the Corners
1.1 Empirical Observations for Psychological/Cognitive Classifications

This activity is strongly supported, and indeed, reflects a fundamental basis for repertory grid approaches. The approach is 'constructivist' in nature and assumes that each problem-solver interacts with the task' target system in a fashion which is described uniquely by the problem-solver.

1.1.1 Interaction: This activity is not addressed but rather the expert's recall of a number of experiences with a number of examples.

1.1.2 Introspection: This activity is very clearly used and supported by rep. grid methods. The expert is required to generate examples (entities) and a class of introspective processes are required for generating attributes (distinctions).

1.1.3 Communication: The expert does little or no communication with the knowledge engineer.

1.1.4 Formalization: The expert, while developing a grid, is asked for distance-based ratings of elements on attribute scales. Although these ratings may be visual rather than numeric, the expert is asked to make relative gradings or ratings. This activity constitutes the requirements for a consistent set of ratings for the elements. Much of the formalization is done by algorithms for use in analyzing the grids.

1.1.5. Explication: This activity is not done in rep. grid methods although automated extensions to the resulting distance-based grid is common.

1.2 Computational/Representational Classifications

1.2.1 Behaviour Data-Base: Behavioural observations of the agent in interaction with the target are not used in rep. grid approaches. However, the results of a series of

agent/target interactions, encapsulated as attribute-value 'cases', can be used to develop a distance-based grid.

1.2.2 Informal Knowledge Base: In strict rep. grid approaches this type of representation is only partially supported due to the rigorous structures of the attribute-value representation. However, extensions (Gaines and Linster, 1990) have linked annotation procedures to rep. grid data.

1.2.3 Formal Knowledge Base: This representation is one of the strengths of the rep. grid approach. The resulting grids are very specific and very well-formed structurally. There is a single form of representation. The grid is open to a variety of analysis procedures which can result in, for example, entailments which can be used for 'stocking' an expert system shell.

1.2.4. Computational Knowledge Base: The resulting grid is not capable, alone, of generating behaviour unless additional procedures are used.

1.3 Methods for Epistemological Classifications

1.3.1 System Identification: Specific behavioural interactions between expert and target are not addressed.

1.3.2 Empirical Abduction: This activity is not supported with rep. grids but require additional procedures which use the grids as a basis for identifying patterns.

1.3.3 Conceptual Analysis: The meaning in rep. grids emerges largely because of the underlying assumptions and mechanisms which produce the patterns of attributes and values. Distinctions and systems of distinctions made by an agent are assumed to underlie the selection of attributes and the rating of entities on these attributes. Further 'meanings' are generated through the use of a variety of analysis procedures, (e.g. principle components analysis, focus, etc.)

1.3.4 Knowledge Replication: A formal, operable replication is not addressed by rep. grids alone and come only after re-formulation for use with inference engines (see Gaines and Linster, 1990)

2. Assessment by Linking Activities

2.1 Plane One (EP to OC): Rep. grid technology and the meaning derived from the grids is heavily based on assumptions about specific cognitive processes. Through distinctions and systems of distinctions (Gaines and Shaw, 1990), an agent is capable of generating a variety of structures upon which grids are based.

2.2 Plane Two (EP to CR): The epistemic notion of 'distinction' is reliably articulated in knowledge representational structures (Gaines and Shaw, 1990). The development of knowledge representation formalisms based on the epistemic primitive of distinction is clearly articulated and formalized.

2.3 Plane Three (OC to CR): A very clear connection is developed between the cognitive process of generating a distinction and the representation of the resulting knowledge forms (concepts, roles, individuals, etc. from Gaines and Shaw, 1990). The resulting rep. grid structures and the methods used to elicit the grid coincide with cognitive processes in form and structure.

3. Assessment of Coherence

3.1 Coherence (OC to CR to EP): Based on the notion of 'distinction', cognitive mechanisms are evident for the belief that such distinction-making mechanisms are used by the agent. The same concept of 'distinction' acts as the basis for the knowledge representation structures used for the artefact. All three justification requirements are evident (face, concept, concurrent). Predictive justification may not be possible because 'out-of-range' input to the artefact cannot easily be incorporated without extension. On the whole, the coherence of the rep. grid method is well established. What is noticeably lacking are validation models for this method.

4. Assessment by Plausible Assumptions

4.1 Frame of Reference: The terms that are used in this method are based on a single concept, that of a 'distinction'. From this core term, a number of others are systematically developed (Gaines and Shaw, 1990). The terms are defined in other contexts and although there may be some differing definitions, the terms, themselves enjoy a good deal of acceptance in the knowledge acquisition community. Extensions to the definitions of the terms are very likely and easily possible due to their clear definitions.

6. Conclusions

6.1 Frame of Reference I and II

The framework presented in this paper is itself embedded in another frame of reference: frame of reference II. Within this frame of reference it is possible to discuss the the parameters of the actual tetrad framework itself. The scope and purpose of frame of reference II is easily described as the world within which the framework presented here can be discussed. Ideally, there would be an objective criterion to decide that the framework presented here has value or 'is better than' another framework. Part of this criterion may be that many or all theories and methods of knowledge acquisition can be assessed by means of the theoretical framework such as the one presented here.

In case there is no objective criteria (as we would predict), the reviewers, as members of a scientific community, would have to have an overlapping understanding with the authors and among themselves of how this framework can be usefully applied to the real world of knowledge acquisition. This enables them to detect aspects of the theoretical framework that can be improved. This can also be seen as that understanding of the theoretical framework including its frame of reference I is a prerequisite to share frame of reference II. In this sense I is imbedded in II.

It can furthermore be seen as an operationalization of shared understanding as social phenomenon: If there is no objective criterion of judgement and judgement still has to be made, social consensus may play this role (which here would mean wide acceptance or rejection of this framework). This seemingly non-scientific step had also to be done in the hard sciences such as physics, when they approached their limits (cf. Heisenberg, 1989). This now allows us to identify three ways, how this (and presumably any) theoretical framework can be challenged:

1. It may fail to describe existing or newly discovered phenomena, (in our case: theories of knowledge acquisition) i.e. the language would not be sufficient enough
2. It may make false predictions about existing or newly discovered phenomena (in our case: theories of knowledge acquisition), i.e.(some of) its conjectures would be refuted

3. It may induce non-operational requirements about relating terms and conjectures to observation, i.e. frame of reference I cannot be established among researchers

Any or all of these may lead for the whole theoretical framework to fail to be socially accepted (in this case) by the scientific community of knowledge acquisition. By challenging the framework presented here, we as a community are developing our understanding within frame of reference II.

6.2 Principled Development of Second Generation Methods and Tools.

The tetrad framework presented in this paper is an attempt to establish a set of parameters for the assessment and comparison of knowledge acquisition methods and tools. It was assumed that those tools and methods were embodiments or artefacts of theories of knowledge acquisition, and implicitly, theories of knowledge. Tools and tool workbenches are considered manifestations of theories of knowledge acquisition. We suggest that beginning a discussion on our framework helps us as a community to establish formal, principled development of future tools and integration of knowledge. The end result is to establish a useful framework for the principled development of second generation knowledge acquisition methods and tools.

6.3 Assessing Existing Knowledge Acquisition Methods

We have presented a framework in which theories of knowledge acquisition may be compared. The framework establishes the 'requirements' of a theory of knowledge acquisition. In this paper, three existing 'theories' of knowledge acquisition were assessed using the tetrad framework. None of those assessed meets all the criteria set out in the framework. We suggest that the framework can help in orderly extension of the methods assessed. It was noted that the set of criteria used for assessment were used only as an example. Depending on the purpose of the assessment or comparison, other criteria may be 'hung' on the hooks of the main parameters of the framework.

6.4 Future Development

We have used the framework to assess three theories and we have identified needed extensions and development to these. The next step is to use the framework to develop new theories of knowledge acquisition and to include in these, machine learning approaches. This would constitute a good test of the framework and encourage frame of reference II discussion. A series of well defined procedural steps involved in the development of a theory is also possible and requires elaboration. Future development can then begin on three fronts: discussion of the framework itself, the use of the framework to help extend the use and range of existing tools, and the development of new theories (methods and tools) for knowledge acquisition.

References

Akkermans, H., van Harmelen, F., Schreiber, G., and Wielinga, B.;
A Formalisation of Knowledge-Level Models for Knowledge Acquisition; accepted for publication in: Int. J. Intelligent Systems.

Althoff, K.D., Maurer, F. and Rehbold, R. Multiple knowledge acquisition strategies in MOLTKE. Wielinga, B., Boose, J., Gaines, B. Schreiber, G., and van Someren, M. (Eds.) **Current Trends in Knowledge Acquisition**, p. 21-40. IOS Press, Amsterdam.

Anjewierden, A., Wielemaker, J. and Toussaint, C. (1990) Shelley- computer aided knowledge engineering. Wielinga, B., Boose, J., Gaines, B. Schreiber, G., and van Someren, M. (Eds.) **Current Trends in Knowledge Acquisition**, p. 41-59. IOS Press, Amsterdam.

Boose, J. H. (1989) Tutorial notes: interactive knowledge acquisition tools. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge Based Systems**. (EKAW'89). Paris,France, (June).

Bradzill, P.B. and Torgo, L. (1990) Knowledge acquisition via knowledge integration. Wielinga, B., Boose, J., Gaines, B. Schreiber, G., and van Someren, M. (Eds.) **Current Trends in Knowledge Acquisition**, p. 90-104. IOS Press, Amsterdam.

Breuker, J. and Wielinga, B. (1987) Use of models in the interpretation of verbal data. In Kidd, A. (ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum.

Clancey, W. (1989) Viewing knowledge bases as qualitative models. **IEEE Expert**, 4(2), 1-23.

Clancey, W. (1990) The frame of reference problem in the design of intelligent machines. In K. vanLehn & A. Newell, **Architectures for Intelligence: The Twenty-Second Carnegie Symposium on Cognition.** Hillsdale:LEA.

Compton, P. and Jansen, B. (1990) A philosophical basis for knowledge acquisition. Boose, J.H. and Gaines, B.R. (Eds.) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Submitted, Banff, November.

Eggen, J., Lundteigen, A.M. and Mehus, M. Integration of knowledge from different knowledge acquisition tools. Wielinga, B., Boose, J., Gaines, B. Schreiber, G., and van Someren, M. (Eds.) **Current Trends in Knowledge Acquisition**, p. 123-142. IOS Press, Amsterdam.

Gaines, B.R. and Shaw, M.L.G. (1990) Cognitive and logical foundations of knowledge acquisition. Boose, J.H. and Gaines, B.R. (Eds.) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. Banff, November.

Gaines, B.R. (1990). An architecture for integrated knowledge acquisition systems. Boose, J.H. and Gaines, B.R. (Eds.) **Proceedings of the Fifth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff, November.

Gaines, B.R. and Linster, M. Development of second generation knowledge acquisition systems. Wielinga, B., Boose, J., Gaines, B. Schreiber, G., and van Someren, M. (Eds.) **Current Trends in Knowledge Acquisition**, p. 143-160. IOS Press, Amsterdam.

Graesser, A.C. and Clark, L.F. **Structures and Procedures of Implicit Knowledge.** New Jersey, Ablex.

Grice, H.P. (1975) **Logic and Conversation Syntax and Semantics.** Speech Acts, Vol. 3., Academic Press.

Heisenberg, W. (1989) **Ordnung der Wirklichkeit:** Munchen, Piper.

Hirst, G. and Regoczei, S. (1989) On 'extracting knowledge from text': modelling the architecture of language users. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge Based Systems.** (EKAW'89). Paris,France, (June).

Johnson, P. (1986) Specification of expertise: knowledge acquisition for expert systems. Boose, J.H. and Gaines, B.R. (Eds.) **Proceedings of the First AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff.

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

Kolodner, J.L. (1983) Maintaining organization in a dynamic, long-term memory. **Cognitive Science**, Vol. 7, 1983.

Newell, A. (1982) The knowledge level. **Artificial Intelligence**, 18, 87-127.

Regoczei, S. and Hirst, G.(1989) Sortal analysis with SORTAL: a software assistant for knowledge acquisition. **Proceedings of the Fourth Knowledge Acquisition Workshop for Knowledge Based Systems.** (KAW'89). Banff, Canada, (Oct).

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

Wetter, T. (1990a) In what sense does knowledge exist ? Alternatives and Consequences for knowledge engineering. In **Proceedings of the Fifth Knowledge Acquisition Workshop for Knowledge Based Systems.** (KAW'90). Banff, Canada, (Nov).

Wetter, T. (1990b) 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**: IOS, Amsterdam.

Wielinga, B., Akkermans, H., Schreiber, G., and Balder, J.(1989) A knowledge acquisition perspective on knowledge-level models. **Proceedings of the Fourth Knowledge Acquisition Workshop for Knowledge Based Systems.** (KAW'89). Banff, Canada, (Oct).

5th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1989.

A Knowledge-Acquisition Approach For Error Handling In Supervised Robotic Systems

Y. Zheng L.K. Daneshmend

McGill Research Centre For Intelligent Machines
McConnell Engineering Building
McGill University, 3480 University Street
Montréal, Québec, CANADA H3A 2A7

Abstract

There has been a significant amount of work in failure-driven approaches that learn or refine planning knowledge to prevent future failures. In contrast, the problem of acquiring error recovery knowledge has received relatively little attention. This paper presents a novel approach which utilizes explanation-based learning (EBL) as a framework for acquisition of error recovery knowledge, and thus reduces the dependency of the autonomous system upon a human operator in recovering from error states. The approach is illustrated by an example of a sensor-based robotic assembly task.

1 Introduction

Supervised autonomous systems which deal with unstructured environments, such as telerobotic systems, cannot plan for every possible eventuality, due to the complexity of the real world. In addition, sensory perception has an associated cost in terms of time and computational effort, and hence on-line sensing can only be utilized to deal with events that have a significant likelihood of occurrence [Miller 87, Boyer 88]. Thus, if an unforeseen or unlikely condition, i.e. an exception, arises, this may result in the system entering an error state from which it cannot achieve the goal of the task it is executing. Under such circumstances, the supervised autonomous system relies upon the human operator to extricate the system from the error, and to enable it to accomplish its task.

The objective of our work is to reduce the dependency of the autonomous system, upon the human operator, in recovering from error states. Thus the problem we are focusing on is that of formulating error recovery strategies, rather than that of the overall planning of complex tasks.

The overall error handling problem can be decomposed into two parts: 1) Exception Analysis. 2) Error recovery. Exception analysis [Taylor 90] is concerned with ascertaining

which exceptional condition caused the task execution to fail. Error recovery is concerned with rectifying the situation such that the system can proceed to a successful completion of the task. Although research on error handling has been conducted for some time [Srinivas 78, Smith and Gini 86], it has not made significant progress. One major difficulty is that replanning for error recovery is computationally expensive [Chapman 87].

Knowledge acquisition is a promising approach to solve error recovery problems. This paper presents an alternative approach to error handling, which utilizes explanation-based learning (EBL) [Dejong and Mooney 86, Mitchell et al. 86] to provide a framework to achieve error recovery. Recovery strategies are learned by using the human operator to carry out the recovery procedure for the first occurrence of an exception. Human experts are skillful at recovering from error states, but they are incredibly poor at producing general deductive rules that account for their behaviour. This accounts for the slowness and inefficiency of acquiring correct error recovery knowledge — knowledge acquisition is a major bottle neck in planning recovery strategies. Consequently, learning from observation is particularly appealing for applications such as telerobotics, where the human operator does not have the time, inclination, or expertise, to explicitly formulate recovery rules. However, designing learning systems is one of the major challenges faced by AI research. Our approach, and its applicability, are illnstrated by an example of a sensor-based robot task.

2 The EBL-Based Error Handling Framework

2.1 Error recovery

Figure 1 is an outline of our error handling framework. It consists of an execution monitor, a recovery module, a learner and a knowledge base which contains planning knowledge and a recovery schema library (RSL). A domain expert (the operator, in the case of a supervised autonomons system) is involved to interact with the system. The input of the execution monitor is the robot-level-program. The execution monitor executes the program, issues commands to the robot, and receives sensor feedback from robot. The history of execution and sensor data are recorded for error recovery and learning.

When an error is detected, it triggers the execution monitor to stop the execution and passes the control to a recovery module. The recovery module tries to find a recovery schema by matching the current error episode against its RSL. If it finds a match, it executes the schema to recover the error. Otherwise, a domain expert will be invoked to give the correct recovery actions. In that case, an opportunity arises for acquisition of error recovery knowledge.

The learner's function is to acquire a recovery schema using EBL techniques. The central concept of this approach is that, after the domain expert carries out a successful recovery, the learner builds an explanation of why the expert's solution successfully solves the problem, generalizes the solution and integrates it into RSL where it will later be used by the system to solve related problems. The success of EBL relies on the correctness and completeness of

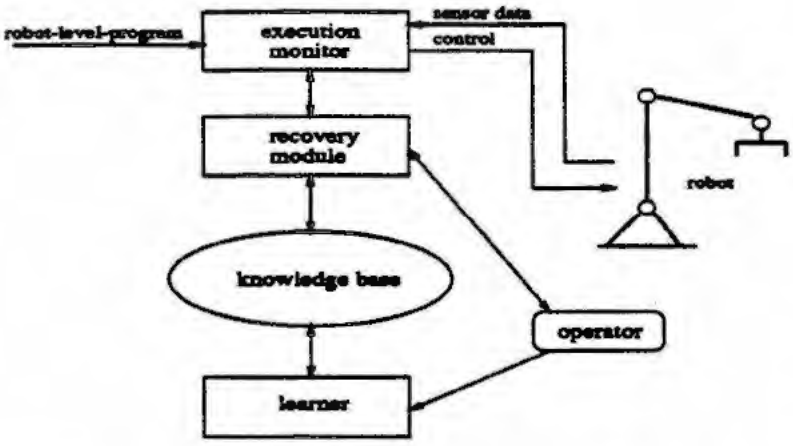


Figure 1: The EBL-based error recovery framework

the domain knowledge [Mitchell et al. 86].

2.2 Expanded explanation-based learning

Fig 2 is an EBL model. Its function can be described as:

Given: Domain knowledge
Training example(s)
A learning concept
An operationality criterion
Determine: An operational description of the learning concept

In fig 2, module 1 takes the training example and the learning concept as its inputs and produces an explanation E_s as output. Module 2 takes the E_s and operationality criterion as inputs and produces a new concept that is operational. Both modules need domain knowledge to accomplish their tasks. This EBL model can be expanded to acquire robot error recovery knowledge. Fig 3 shows the expanded EBL model. It functions as follows

Given: Domain knowledge
A training example of error recovery
An operationality criterion
Determine: An operational error recovery schema

The parser and generalizer in figure 3 function as module 1 and module 2 respectively. But there is a major improvement in the expanded learning model in comparison with the one on fig 2. In fig 2 we need to input a learning concept explicitly [Mitchell et al. 86]. In fig 3, the learning concept is no longer needed. The error analyzer takes the error state from

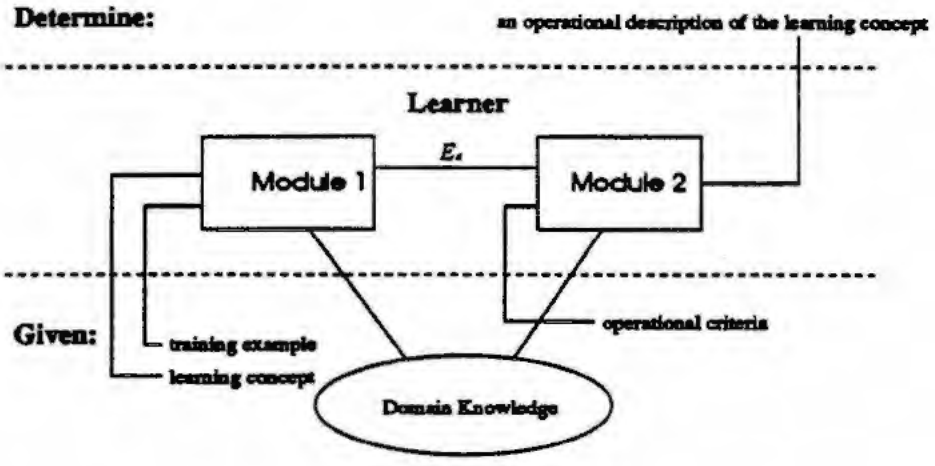


Figure 2: The EBL model

the training example together with domain knowledge to derive a learning concept. This is significant because it is impractical to impart the learning concept before hand. Learning is a dynamic process. The learning system should derive what it want to learn and to guide its learning process. Here we assume the domain knowledge contains a set of concept rules which are all non-operational. The error analyzer will find out which one should be chosen to guide the learning process.

2.3 Objectives

Our efforts will focus on developing the techniques and algorithms to implement the learner. The following issues are addressed in this paper:

1. Analyze the error source.
2. Construct a causal model which explains how the recovery actions can achieve the goal from the error state.
3. Generalize the recovery strategy.

Some other issues are considered as future research topics:

1. Integrate a new schema into the RSL.
2. Deal with incomplete domain knowledge.
3. Evaluate whether a strategy is worth learning.
4. How to refine a schema if it fails.

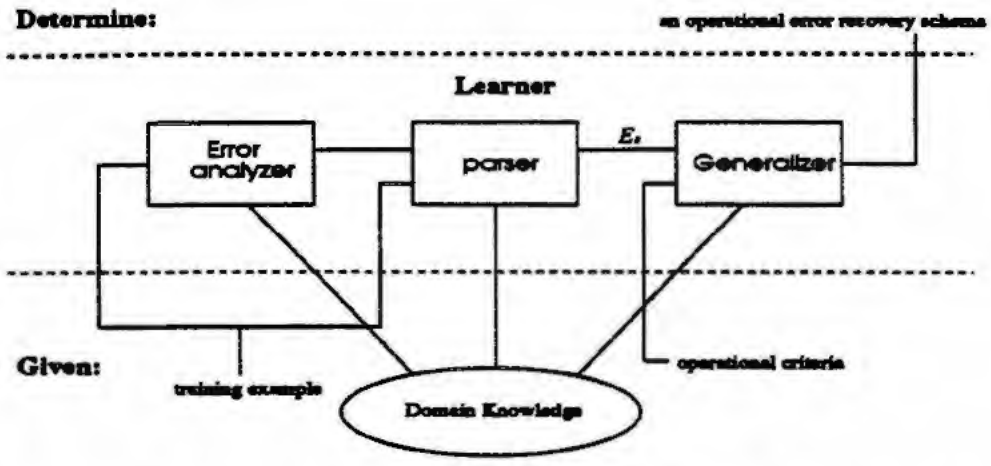


Figure 3: The modified learning model for acquiring error recovery knowledge

3 Error Model And Error Analysis

3.1 An error scenario

Our approach can be illustrated by a straight forward assembly task — inserting a peg into a hole. The object with the hole is placed at P which is at a position where the robot is close to a kinematic singularity. The gripper grasps the Peg and moves it above the Hole, then inserts the Peg downward. Since the accuracy of the robot's motion is degraded near a singularity, the Peg collides with the Hole. Fig 4 illustrates the error state. A collision signal indicates an error and triggers the recovery module to recover from the error. If the recovery module fails to find an appropriate schema to correct the error, a human operator is involved to provide the recovery actions.

Observing the hole was placed in a position where, when performing an insertion, the robot was near a singularity, the operator knows this resulted in an inaccurate insertion that accounts for the collision. The operator successfully recovers the error by relocating the hole in a better position and redoing the insertion. Here are his recovery actions:

put peg at P_1
pick hole at P
put hole at P_2
pick peg at P_1
move peg above hole
insert peg down

The learner will take the error scenario and recovery actions as a training example in its learning process. The learner consists of an error analyzer, a parser and a generalizer. It

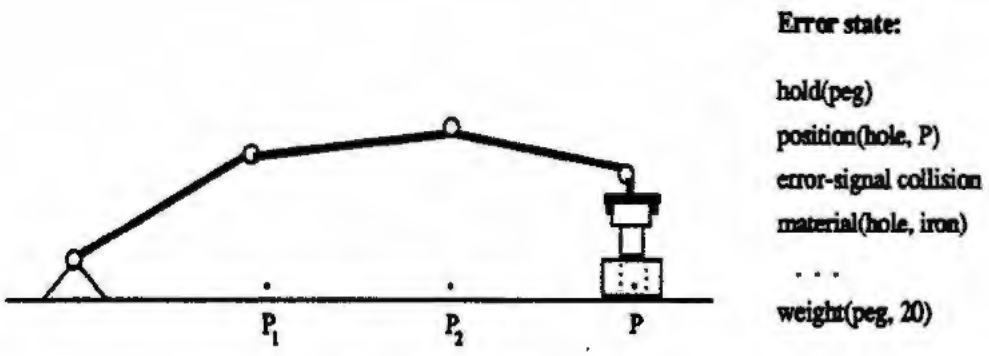


Figure 4: Error state of the given example.

incrementally augments the recovery schema library with a new schema. This is achieved via a three stage process of error analysis, explanation, and generalization.

3.2 The error model

The ultimate goal of the error analysis process is to determine the cause of failure, and hence to be able to retrieve a corresponding concept rule that describes, in a non-operational manner, what can be done about the failure. The consequent of the concept rule is used as a learning concept to guide the construction of an explanation of why those recovery actions can correct the error. This explanation will help to abstract a recovery schema.

Some assumptions are made in our discussion. First, we are only interested in learning error recovery knowledge for those errors that are recoverable. Some errors, such as motor failure, robot gripper broken etc., can only be recovered from by the human operator and there is no point in learning how to recover from these. Second, we consider the failures that are caused by a single error source: multiple error sources will make error analysis intractable. Third, we only focus on learning error recovery knowledge for local errors. A local error means the error is detected immediately after execution of an action. Furthermore, we assume we have adequate sensor information and domain knowledge to derive the most probable cause of error.

The technique for error analysis discussed in this section is based on an error model and an error analysis technique. The error model is the knowledge about failures — why they occur, and what can be done about them. The error model consists of two parts:

1. **A set of action-cause-feature models:** An action-cause-feature model (ACF) is a $5-tuple$ $\langle a_k, C, F, P, M \rangle$. Here a_k represents a specific action. Vector $C = (c_1, c_2, \ldots c_n)$ represents the n possible failure causes. Vector $F = (f_1, f_2, \ldots f_m)$ represents the m relevant features that can be sensed by robot monitor. The value of f_i $(i = 1, 2, \ldots m)$ is defined as:

a_k — insert peg into hole

$C = (c_1, c_2, c_3, c_4, c_5)$

$F = (f_1, f_2, f_3, f_4, f_5, f_6)$

$P = (0.40, 0.10, 0.20, 0.20, 0.10)$

$$M = \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{bmatrix} 0.85 & 0.80 & 0.90 & 0.90 & 0.30 \\ 0.30 & 0.40 & 0.60 & 0.40 & 0.99 \\ 0.05 & 0.95 & 0.01 & 0.01 & 0.20 \\ 0.01 & 0.01 & 0.98 & 0.01 & 0.20 \\ 0.01 & 0.01 & 0.01 & 0.98 & 0.15 \\ 0.70 & 0.10 & 0.01 & 0.01 & 0.40 \end{bmatrix} \end{array}$$

c_1 — action inaccurate
c_2 — near singularity
c_3 — defective peg
c_4 — defective hole
c_5 — drop peg

f_1 — collision
f_2 — hand empty
f_3 — hand near singular location
f_4 — flaw on peg
f_5 — flaw on hole
f_6 — vacuum too low

Figure 5: The action-cause-feature model

$$f_i = \begin{cases} 1 & \text{if } f_i \text{ is sensed} \\ 0 & \text{otherwise} \end{cases}$$

Vector $P = (p(c_1), p(c_2), \ldots p(c_n))$ is the a priori probabilities for $c_1, c_2, \ldots c_n$ under action a_k. We call M a cause-feature matrix which represents the conditional probabilities for each feature f_i under condition c_j. i.e. the element m_{ij} has the value $m_{ij} = p(f_i \mid c_j)$. M is a $m \times n$ matrix where m rows correspond to the m features and n columns correspond to the n failure causes. For each action, there exists an action-cause-feature model stored in the knowledge base. Take an action *"insert peg into hole"* as an example. Fig 5 shows its action-cause-feature model.

2. **A set of concept rules:** Concept rules provide brief ideas about how to correct a particular kind of failure. For example, Fig 6 lists some concept rules that deal with singularity location, defective hole, defective peg and inaccurate action respectively. Each concept rule has the form:

If (and (action: ⟨ failure action a_k ⟩)
(signal: ⟨ failure signal b_k ⟩)
(cause: ⟨ failure cause c_k ⟩))
Then (recovery: ⟨ recovery strategy d_k ⟩)

The antecedent of the rule is a conjunction of three simple propositions which describe the failure action, failure signal and failure cause. Each of them are operational to the system. The consequent of the rule contains a recovery strategy: this is called a learning concept. To recover from a failure is quite a difficult task [Smith and Gini 86]. Generally speaking, we need to undo certain actions, achieve some required conditions, then redo the undone steps. Knowledge about what needs to be undone, and what needs to be redone, for a particular failure cause is extremely complicated and domain dependent for any realistic robot world. The recovery strategies of concept rules do not contain this procedural knowledge. Their knowledge to deal with errors is non-operational (i.e. non-executable by the system). Obviously human operators are skillful in producing non-operational rules. However, they are very poor in producing operational ones. Our objective is to transfer the learning concepts into operational knowledge by the learning process described below.

3.3 Deriving the most likely failure cause

When an erroneous action a_k is determined from the failure signal, its corresponding ACF model $\langle a_k, C, F, P, M \rangle$ will be used to derive the most likely failure cause. We first query F from sensor data to see what features have been detected. Given the occurrence of an error signal, we know that it will have been triggered by at least one feature. Based on the number of features sensed, the postpriori probabilities can be calculated as follows:

Case 1: Only one feature f_i is sensed. In this case, the postpriori probability vector ($p(c_1 \mid f_i)$, $p(c_2 \mid f_i), \ldots, p(c_n \mid f_i)$) can be calculated by Bayes's theorem

$$p(c_k \mid f_i) = \frac{p(f_i \mid c_k)p(c_k)}{\sum_{j=1}^{n} p(f_i \mid c_j)p(c_j)} \tag{1}$$

here the $p(f_i \mid c_k)$ and $p(c_k)$ are conditional probability and a priori probability. They are all known from the ACF model.

Case 2: Equation 1 can be expanded if more than one feature is detected simultaneously, provided that these features are independent from each other. Without loss of generality, assume feature $f_1, f_2, \ldots, f_k$ are detected. The postpriori probability vector ($p(c_1 \mid f_1 f_2 \ldots f_k)$, $p(c_2 \mid f_1 f_2 \ldots f_k), \ldots, p(c_n \mid f_1 f_2 \ldots f_k)$) can be calculated as

$$p(c_i \mid f_1 f_2 \ldots f_k) = \frac{p(f_1 \mid c_i)p(f_2 \mid c_i) \ldots p(f_k \mid c_i)p(c_i)}{\sum_{j=1}^{n} p(f_1 \mid c_j)p(f_2 \mid c_j) \ldots p(f_k \mid c_j)p(c_j)} \tag{2}$$

If (and (action: insert peg into hole)
(signal: collision)
(cause: singularity)
Then (recovery: IPIH at NSL)

If (and (action: insert peg into hole)
(signal: collision)
(cause: defective peg)
Then (recovery: change peg)

If (and (action: insert peg into hole)
(signal: collision)
(cause: defective hole)
Then (recovery: change hole)

If (and (action: insert peg into hole)
(signal: collision)
(cause: action inaccurate)
Then (recovery: re-align peg to hole)

IPIH — insert peg into hole
NSL — nonsingular location

Figure 6: Some concept rules

Example: Given an action "insert peg into hole" and the ACF model shown in figure 5, suppose features f_1 and f_3 are detected. From equation 2, the postpriori probabilities are

$$p(c_i \mid f_1 f_3) = \frac{p(f_1 \mid c_i)p(f_3 \mid c_i)p(c_i)}{\sum_{j=1}^{n} p(f_1 \mid c_j)p(f_3 \mid c_j)p(c_j)}$$

substitute the value in F and M, we have

$$p(c_1 \mid f_1 f_3) = 0.16$$

$$p(c_2 \mid f_1 f_3) = 0.74$$

$$p(c_3 \mid f_1 f_3) = 0.02$$

$$p(c_4 \mid f_1 f_3) = 0.02$$

$$p(c_5 \mid f_1 f_3) = 0.01$$

So the most possible failure cause is c_2, i.e. proximity to a kinematic singularity.

The success of error analysis is based on the error model. Since the error model is a simplified model of the real world, the error analysis cannot guarantee correct results for those errors which are not included in the error model or not correctly described in the error model. Generally speaking, for complicated robot environments, error analysis is a credit-assignment problem .

3.4 Acquiring learning concepts

The most probable cause of failure, the erroneous action and the error signal will help to retrieve a concept rule whose consequent contains a recovery strategy. The recovery strategy will be used as a learning concept.

In the above example the failure-cause is location of the object near a singularity; the failure-action is insertion of the peg into the hole; and the failure-signal is collision. The concept rule

```
If (and (action: insert peg into hole)
        (signal: collision)
        (cause: singularity)
Then    (recovery: IPIH at NSL)
```

is retrieved. Therefore the learning concept is "insert peg into hole at nonsingular location".

4 Building An Explanation

4.1 Building the causal model

The sequence of recovery actions $A = a_1, a_2, \ldots, a_n$ implies the intention of the domain expert to achieve the goal from the failure state. This intention should fit the learning concept if the error analysis derives it correctly. The explanation process builds causal relations between the learning concept and the recovery actions $a_1, a_2, \ldots, a_n$. These causal relations veriify that the sequence of actions $a_1, a_2, \ldots, a_n$ is a special example of the learning concept.

The parser takes the learning concept, the domain knowledge as well as the recovery actions $a_1, a_2, \ldots, a_n$ as its inputs and produces a derivation tree, E_s, as its output (ref. figure 3). Since E_s reflects the causal relations between the learning concept and the recovery actions, we call it the causal model which will be used later on for generalization process.

There are basically two ways to derive the tree: bottom-up and top-down. The bottom-up method [Segre 88] starts from the recovery actions, gradually uses higher level operators to describe the context in which the lower level operators are occurring. As the levels go up, the descriptions become more and more abstract. Eventually the description at the top level matches the learning concept.

The parser in our approach adopts the top-down method. It starts from the root node that contains the learning concept as its task. We expand the root node hy using a collection of operators to reduce the task to a new set of subtasks, and each of these subtasks can, itself, be further reduced to a set of subsubtasks ... etc.; ultimately "bottoming out" with leaf nodes. Each leaf node contains either a recovery action a_i or a condition that is verified by the facts in database.

The database is used to hold facts and relations about the past and current robot world. Facts and relations that change with time are recorded with time stamps. All requests by the learner for information about whether a certain condition was true at some specific time must be routed through the database.

There are two kinds of leaf nodes in our top down derivation. The ones containing recovery actions are called action nodes. The ones containing conditions are called condition nodes. All the action nodes are ordered from left to right according their position in the tree. Each action node has a time stamp associated with it. For example, let $N_1, N_2, \ldots, N_n$ be action nodes ordered from left to right. Assume each action takes one time unit. Then the time stamps for these nodes are $t(N_1) = 1, t(N_2) = 2, \ldots, t(N_n) = n$ respectively. Each action node N_i contains an action b_i. So the action sequence $b_1, b_2, \ldots, b_n$ is contained in $N_1, N_2, \ldots, N_n$. All the condition nodes are verified by the facts in the database as follows:

- The condition nodes priori to N_1 are verified by the facts with time stamp $t = 0$;
- The condition nodes between N_i and N_{i+1} are verified by the facts with time stamp $t = i$;
- The condition nodes after N_n are verified by the facts with time stamp $t = n$.

If the action sequence $b_1, b_2, \ldots, b_n$ contained by $N_1, N_2, \ldots, N_n$ matches the recovery actions $a_1, a_2, \ldots, a_n$ given by human operator, then there exists at least one parse tree from which the recovery actions can be derived from the learning concept. For each particular parse tree, those recovery actions has a unique leftmost derivations. In the next section, an algorithm for leftmost derivation will be described in detail.

Building the explanation is critical in EBL as it has the potential for combinatorial explosion — an intractable theory problem [Mitchell et al. 86]. Thus far there is no satisfactory, complete, solution for this problem. Effectively constructing the explanation is still a research field with some unsolved problems.

Training example

Error state:
- hold(peg)
- position(hole, P)
- error-signal collision
- material(hole, iron)
- ...
- weight(peg,20)

Recovery actions:
- put peg at P_1
- pick hole at P
- put hole at P_2
- pick peg at P_1
- move peg above hole
- insert peg down

Domain knowledge

IPIH — insert peg into hole
NSL — nonsingular location

R1: If (and (relocate hole at NSL)
(align peg above hole)
(insert peg down))
Then (IPIH at NSL)

R2: If (and (get ?X)
(nonsingularity(?P))
(freespace(?P)
(put ?X at ?P))
Then (relocate ?X at NSL)

R3: If (and (free hand)
(position(?X, ?P))
(pick ?X at ?P))
Then (get ?X)

R4: If (and (hold(?X))
(freespace(?P))
(put ?X at ?P))
Then (free hand)

R5: If (and (position(?X, ?P))
(pick ?X at ?P)
(move ?X above ?Y))
Then (align ?X above ?Y)

R6: If (put ?X at ?P)
Then (position(?X, ?P))

R7: If (and (position(?X, ?P))
(move ?Y above ?P))
Then (move ?Y above ?X)

Operationality criterion

Some operational expressions:

- hold(?X)
- freespace(?P)
- put ?X at ?P
- position(?X, ?P)
- pick ?X at ?P
- nonsingularity(?P)
- move ?X above ?P
- insert peg down

Figure 7: The necessary inputs for a learner

4.2 The leftmost derivation algorithm

Before describing the algorithm for the leftmost derivation, we introduce some notations used in the algorithm. Let $a_1, a_2, ...a_n$ represent the recovery actions. The kth recovery action refers the recovery action a_k. Let T_i, T_j, T_k... represent intermediate derivation trees. For each tree, say T_i, there is a corresponding number t_i associated with it. t_i has two functions: i) It indicates which recovery action has been explained and which one should be explained next. ii) It indicates the time stamp for checking facts in the database. A STACK is used to store all possible intermediate leftmost derivations during explanation process.

Algorithm

Step 1, Build an initial tree T_i with the learning concept as the root node of the tree. Mark the root node as a open node. Set the corresponding number $t_i = 0$. Push the T_i into STACK.

Step 2, If the STACK is empty, stop the process and signal the failure of explanation building.

Step 3, Pop a tree, say T_j, from STACK. If all the nodes of T_j are closed, then consider two cases:

- If the t_jth recovery action is the last recovery action, then we have found the leftmost derivation tree for $a_1, a_2, ...a_n$. Stop the process and signal success.
- If the t_jth recovery action is not the last recovery action, then discard T_j, and go to step 2.

Step 4, If the leftmost open node of T_j matches the $(t_j + 1)$th recovery action, then mark the node a closed action node and set $t_j = t_j + 1$. Push T_j into STACK. Go to step 2.

Step 5, If the leftmost open node of T_j is verified by a fact with time stamp $t = t_j$ in database, then mark the node as a closed condition node. Push T_j into STACK. Go to step 2.

Step 6, Expand the leftmost open node of T_j, say N_l, as follows:

- Case1: If there is only one rule, say R_m, which can expands N_l, i.e. N_l matches the right side of R_m, then each conjunction of the left side of R_m becomes a child of N_l. Mark N_l as closed node and mark all its children as open nodes. Push T_j into STACK. Go to step 2.
- Case2: If there is more than one rule which can expand N_l, without loss of generality, say $R_1, R_2, ...R_k$ can expand the N_l, then we first duplicate T_j k times, so we have $T_{j_1}, T_{j_2}, \ldots T_{j_k}$. For each T_{j_i} ($i = 1, 2, \ldots, k$) we use R_i to expand it as in Case1. After expanding the leftmost node N_l for each tree, push $T_{j_1}, T_{j_2}, \ldots, T_{j_k}$ into STACk. Go to step 2.

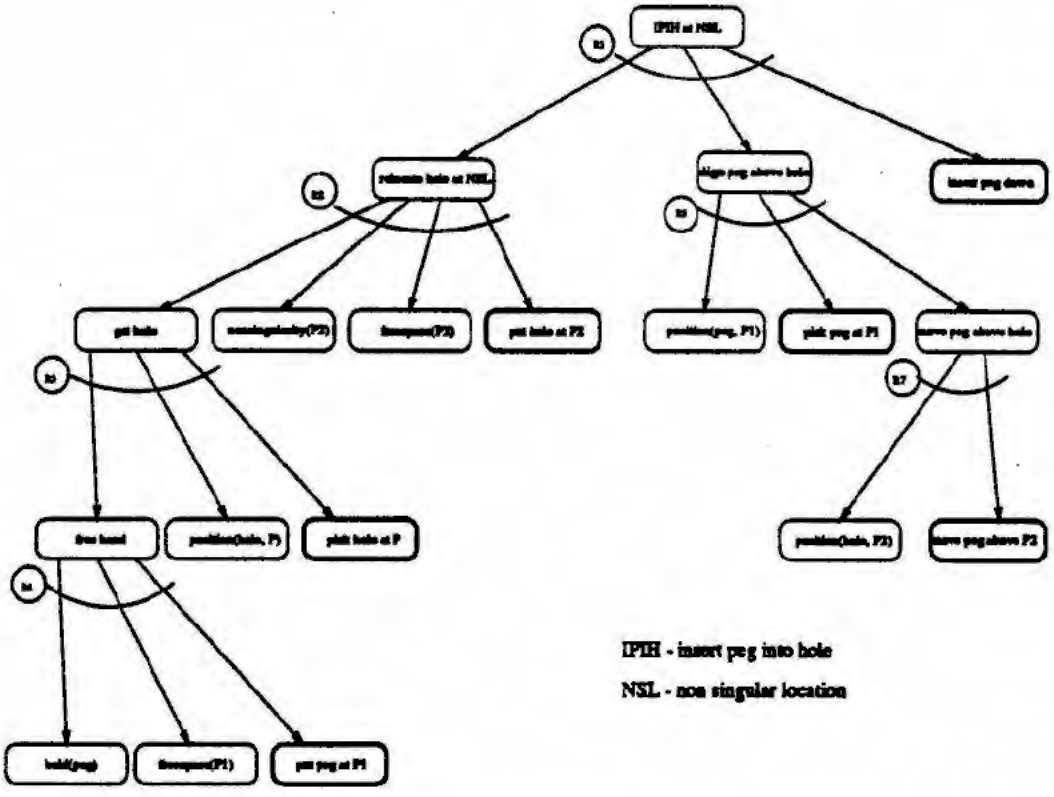


Figure 8: The derivation tree of the given example

Step 7, Discard the T_j. Go to step 2.

It is possible that some recovery actions have more than one parse tree, i.e. there is more than one leftmost derivation. In the above algorithm we are only interested in finding the first leftmost derivation. We don't care if other leftmost derivations exist.

Fig 8 shows how the causal model is constructed for the previous example. The training example and some relevant domain knowledge are given in figure 7. In the previous section, we derived the learning concept — "IPIH at NSL". The parser takes these as its inputs and constructs a derivation tree by using the leftmost derivation algorithm. In figure 8, each box represents a tree node. The small circles with rule names in them are the corresponding rules. The leaf nodes of the tree are divided into two classes. The ones with bold face boxes are action nodes. The rest of the leaf nodes are condition nodes. After successfully building the causal model the generalization process follows.

5 The Generalization Process

Explanation-based learning depends on having an explanation on which to base generalization. The explanation constructed in section 4 has separated the relevant feature values in

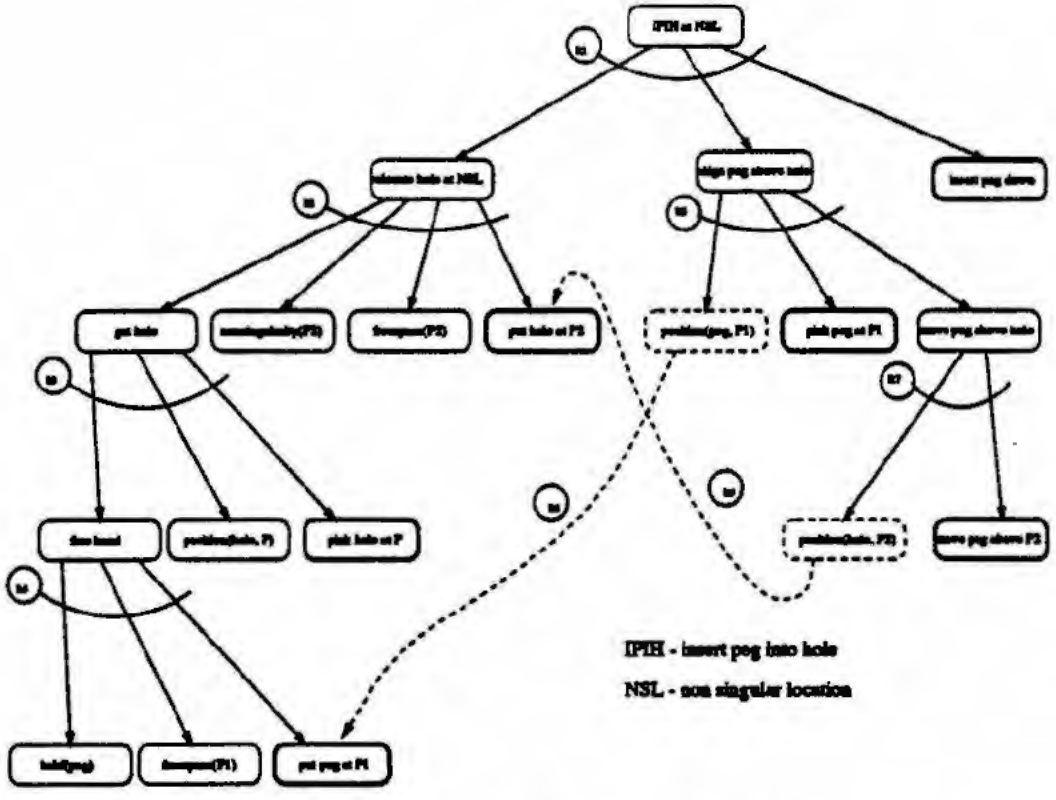


Figure 9: Find out the dependent conditions

the example from the irrelevant ones. For example, the facts "hold(peg)" and "position(hole, P)" in the error state, are relevant. They appear in the condition nodes of derivation tree and will be used in deriving new schema. But the "weight(peg, 20)", "material(hole, iron)" are irrelevant, therefore they are discarded. The generalization process analyzes the explanation to determine the particular constraints on those feature values that are sufficient for the explanation structure to apply in general.

The generalization process includes three steps. First, it analyzes the condition nodes in derivation tree. As we mentioned earlier, all condition nodes are verified by facts with appropriate time stamps. But we are only interested in those condition nodes that contain the necessary conditions for applying the sequence of recovery actions. For example, "hold(peg)" is a necessary condition for the recovery action sequence. It has to be true before the action "put peg at P_1" is executed. But "position(peg, P_1)" is not a necessary condition for the recovery action sequence. This condition should be true before the action "pick peg at P_1", but after action "put peg at P_1" the condition become valid. So this condition is action dependent. We call it a dependent condition. Similarly, the condition "position(hole, P_2)" is also a dependent condition that depends on the action "put hole at P_2".

We can easily identify the dependent conditions by checking their validation at different times. Furthermore, it is an easy task to find out to which actions they depend on. In figure 9 the condition nodes with dash boxs are dependent conditions. Dash arrows are used

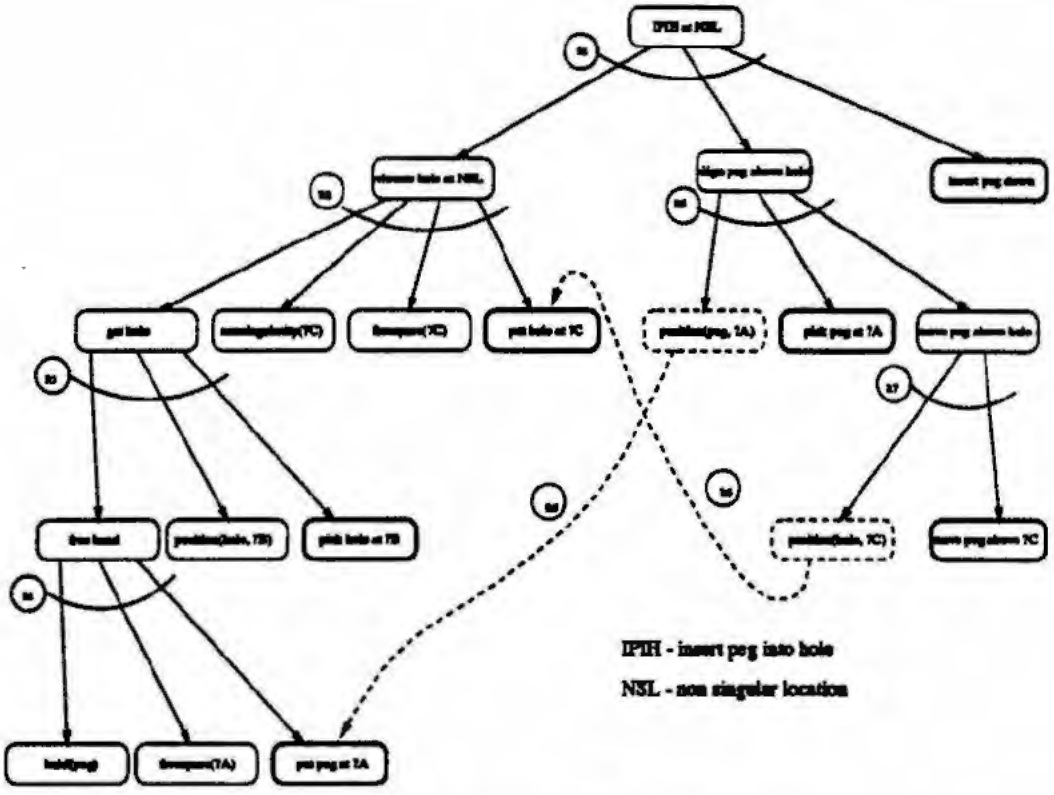


Figure 10: The generalized explanation

to indicate to which actions they depend on. In the following discussion, the dependent conditions are no longer considered as necessary conditions.

The second step generalizes the explanation by replacing constants with variables. To ensure a valid generalization of the training example, the goal regression method [Waldinger 77] is used. This involves regressing the learning concept through the explanation structure. During traversing the tree from top to bottom relevant constants are determined and are replaced by proper variables. The result is a more generalized explanation. Figure 10 is resulted from figure 9 after above process. It illustrates how constants are replaced by variables.

In the last step, the operationality criterion is applied to prune the tree. Beginning from the leaf nodes we proceed in a bottom-up manner, and inspect those nodes with operational expressions. If a parent node and all of its children contain operational expressions, the children are pruned so that the parent node becomes a new leaf node. If the pruned child nodes are all condition nodes we mark the new leaf node, which had been the parent node, as a condition node. Otherwise we mark it as an action node. This process will simplify the tree, resulting in more abstract expressions for its leaf (operational) nodes. The operationality criterion of our example is given in figure 7.

Finally, from the generalized explanation, together with the concept rule derived from error analysis, we can construct a new schema. The new schema reflects a general strategy for

achieving the recovery. The operational definition of the new schema has two parts. One describes the error state. The other lists the sequence of recovery actions. The former is formed by combining the conjunctions of condition nodes together with the antecedent of the concept rule. The latter is formed from the actions in the action nodes. Applying the above procedure to the given example, a new recovery schema is learned:

```
{Recovery Schema
        (Error-state: (action: insert peg into hole)
                      (signal: collision)
                      (cause: singularity)
                      (hold(peg))
                      (position(hole, ?B))
                      (freespace(?A))
                      (freespace(?C))
                      (nonsingularity(?C)))
        (Actions:     (put peg at ?A)
                      (pick bole at ?B)
                      (put hole at ?C)
                      (pick peg at ?A)
                      (move peg above ?C)
                      (insert peg down )) }
```

The new schema is integrated into the RSL and will be used in the future to solve related failures.

6 Comparison With Previous And Related Work

Recently, learning techniques have become a popular approach to deal with errors and exceptions. Failure-driven approaches to refine planning knowledge have attracted great attention. [Hayes-Roth 83, Winston 86] addressed how to learn incrementally using similarity-based techniques. These approach use ad hoc heuristics and many training examples as inputs to guide when and how to refine the knowledge.

More applicable to our research are EBL approaches [Segre 88, Hammond 87, Gupta 87, Mostow and Bhatnagar 87, Minton at al. 89]. Segre's work [Segre 88] on ARMS has a strong influence on our research. ARMS primarily focuses on developing a robot problem solver that generated planning schemata by observing positive solutions. Assuming complete domain knowledge, an explanation process followed by generalization achieved the learning of assembly plans.

Work by [Hammond 87] on the CHEF system also addresses the problem of refining existing plans. When a plan fails, CHEF first explains why the plan failed. CHEF then uses this explanation to classify the conditions that led to the failure. This classification suggests

corresponding actions to correct the failure. CHEF also creates a rule using the generalized condition that predicts the failure.

Gupta's work on TRAP [Gupta 87] addresses improving planning knowledge through failure. EBL is used to explain the cause of failure and to learn a new constraint. This constraint is added as another field in operator definition and used as a guard to prevent future failure.

PRODIGY [Minton at al. 89] and FAILSAFE [Mostow and Bhatnagar 87] are designed to acquire search control rules that are effective in reducing total search time for task domains. These rules can be learned either from failure [Mostow and Bhatnagar 87] or from successful problem solving decisions and unforeseen goal interactions [Minton at al. 89].

The most obvious differences, between our approach and previous EBL methods for failure driven learning, are the follows:

- Our learning algorithm does not need a learning concept that is explicitly provided as an input to the learner. Instead, the algorithm itself finds an appropriate learning concept to guide the learning. This is achieved by the error analysis process.
- Previous failure-driven approaches [Hammond 87, Chien 87, Gupta 87, Winston 86, Minton at al. 89, Mostow and Bhatnagar 87, Birnbaum et al. 89, Hayes-Roth 83] focused on learning or refining their planning knowledge to *prevent* a future failure. Our approach is aimed at learning error recovery schemata that will be used to *recover* from future failures. This is an unique characteristic of our method. The learned schemata are integrated into the RSL, which is off-line and is only invoked when an error state occurs. This scheme will keep on-line planning knowledge as efficient as possible.

7 Discussion

Most EBL problem solvers learn by analyzing why a solution succeeds in solving a problem [Minton at al. 89, Segre 88]. The result is a knowledge structure (such as schema or macro-operator) that describes how to solve similar problems should they arise in future. However, EBL can also be used to learn why a problem-solving method failed [Chien 87, Gupta 87, Hammond 87, Mostow and Bhatnagar 87]. By explaining why a method failed to solve a problem, a system can improve its problem solving knowledge. There exist two paradigms in using EBL to deal with robot errors. The first approach aims at perfecting a robot planner to produce a "perfect" plans by learning planning knowledge. The second approach focuses on knowing how to recover from the error after it occurs.

There is, of course, a trade off between acquiring planning knowledge and and acquiring error recovery knowledge. For those errors with a significant likelihood of occurrence, building a planner that learns is preferable. For those errors which occur occasionally, building a system that learns failure recovery knowledge is more efficient.

In this paper we have presented a technique to acquire error recovery knowledge for local errors. A more practical and more difficult problem is to learn recovery knowledge for interactive errors. In such a case we would have to assume that some, albeit limited, sensor information is available: i.e. the robot explicitly checks for certain conditions to detect failures. The execution monitoring is pragmatic in the sense that only those conditions that are easy to check on the basis of information available directly from feedback, sensors, etc., are tested. A consequence of this lack of comprehensiveness is that failures can propagate and may be detected only at a subsequent step. Learning to recover these kinds of errors is more challenging, and will be part of our future research efforts. We are presently working towards implementation of our error recovery scheme and its demonstration in the laboratory.

Acknowledgements

This work has been partially supported by the Province of Québec's MESST Action Structurantes program, hy a Federal Government NSERC operating grant, and by Project C-2 of the Canadian Institute for Robotics and Intelligent Systems.

References

[Birnbaum et al. 89] Birnbaum L., Collins G. and Krulwich B., *Issues in the justification-based diagnosis of planning failures*, Proceedings of The 6th International Workshop on Machine Learning, p194-196, 1989.

[Boyer 88] Boyer, M., *A Knowledge-Based System for On-Line Robot Error Recovery*, M.Eng. Thesis, Department of Electrical Engineering, McGill University, Montréal, Canada, July 1988.

[Chapman 87] Chapman D., *Planning for conjunctive goals*, Artificial Intelligence, 32, 3(1987) pp.333-378.

[Chien 87] Chien S.A., *Extending explanation-based learning: Failure-driven schema refinement*, Proc. Third IEEE Conf. on Artificial Intelligence Applications, 1987, pp.106-111.

[Dejong and Mooney 86] Dejong G. and Mooney R., *Explanation-based learning: An alternative view*, Machine Learning 1, 1986, pp.145-176.

[Gupta 87] Gupta A., *Explanation-based failure recovery*, Proceedings of AAAI Conference, 1987, pp.606-610.

[Hall 88] Hall R., *Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains*, Machine Learning 3, 1988, pp.45-77.

[Hammond 87] Hammond K.J., *Explaining and repairing plans that fail*,, Proceedings of the International Joint Conference on AI, 1987, pp.109-144.

[Hayes-Roth 83] Hayes-Roth, F., *Using proofs and refutations to learn from experience*, Machine Learning: Volume 1, Morgan-Kaufmann Publishers, 1983, pp.221-240.

[Miller 87] Miller W. T., *Sensor-based Control of Robotic Manipulators Using a General Learning Algorithm,* IEEE Journal of Robotics and Automation, vol. RA-3, No. 2, April 1987. pp.157-165.

[Minton at al. 89] Minton S., et al, *Explanation-based learning: A problem solving perspective,* Artificial Intelligence 40 (1989), pp.63-118.

[Mitchell et al. 86] Mitchell T., Keller R.M. and Kedar-Cabelli S., *Explanation-based generalization: A unifying view,* Machine Learning 1, 1986, pp.47-80.

[Mostow and Bhatnagar 87] Mostow J. and Bhatnagar N., *Failsafe — A floor planner that use EBG to learn from its failures,* Proceedings of the International Joint Conference on AI, 1987.

[Segre 88] Segre, *Machine learning of robot vassembly plans,* Kluwer Academic Publishers, 1988.

[Smith and Gini 86] Smith R.E and Gini M. *Reliable real-time robot operation employing intelligent forward recovery,* Journal of robotic systems 3(3), 1986, pp.281-300.

[Srinivas 78] Srinivas S. *Error recovery in robots through failure reason analysis,* Proc. AFIPS National Computer Conf., 1978, pp.275-282.

[Taylor 90] Taylor P.M., Halleron I., and Song X.K., *The application of a Dynamic Error framework to robotic assembly,* Proceedings IEEE International Conference on Robotic and Automation, 1990, pp.170-175.

[Waldinger 77] Waldinger R., *Achieving several goals simultaneously.* Readings in Artificial Intelligence N. Nilsson and B. Webber (Eds.), Tioga, Palo Alto, CA, 1981, pp.250-271.

[Winston 86] Winston P. H., *Learning by augmenting rules and accumulating censors,* Machine Learning, Volume 2, Morgan-Kaufmann Publishers, 1986, pp.45-61.