Proceedings of the 3RD KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre
Banff, Alberta, Canada
November 6-11, 1988

Sponsored by the
American Association for
Artificial Intelligence

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

Proceedings of the
3RD KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre, Banff, Alberta, Canada
November 6-11, 1988

Sponsored by 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.

3RD 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 viii

Technical Papers (listed alphabetically by first author)

A Framework for the Validation of Knowledge Acquisition 1-1
Izak Benbasat, Jasbir S. Dhaliwal, University of British Columbia

Automated Knowledge Acquisition for a Computer Hardware Synthesis System 2-1
William P. Birmingham (University of Michigan), Daniel P. Siewiorek (Carnegie Mellon University)

A Survey of Knowledge Acquisition Techniques and Tools 3-1
John H. Boose, Boeing Advanced Technology Center

How to Do With Grids What People Say You Can't 4-1
Jeffrey M. Bradshaw, John H. Boose, Stanley P. Covington, Peter J. Russo, Boeing Advanced Technology Center

A Methodology for Validating Large Knowledge Bases 5-1
Rajiv Enand, Gary S. Kahn (Carnegie Group, Inc.), Robert A. Mills (Ford Motor Company)

Knowledge Elicitation in Theory and Practice: An Integrated Program of Research and Training 6-1
Diana E. Forsythe, Bruce G. Buchanan, Stanford University

Synthesizing an Effective Control Strategy from Acquired Domain Knowledge 7-1
Michael J. Freiling, Tektronix Laboratories

Knowledge Acquisition and Technology 8-1
Brian R. Gaines, University of Calgary

An Interactive Performance Based Knowledge Acquisition Methodology 9-1
William A. Gale, AT&T Bell Laboratories

A Method for Acquiring Strategic Knowledge 10-1
Thomas Gruber, Stanford University

Learning Importance of Concepts: Construction of Representative Network 11-1
Ken-ichi Handa, Shun Ishizaki, ElectroTechnical Laboratory, Japan

Knowledge Acquisition Planning 12-1
Lawrence Hunter, Yale University

Knowledge Representation for Knowledge Elicitation 13-1
Nancy E. Johnson, C. M. Tomlinson, Brunel University

Progress in Automated Knowledge Acquisition Tools: How Close Are We to Replacing the Knowledge Engineer? 14-1
Catherine M. Kitto, Boeing Advanced Technology Center

Building Expert Systems with KNACK 15-1
Georg Klinker (Digital Equipment Corp.), Casey Boyd (Carnegie Mellon University), David Dong (Digital Equipment Corp.), Jon Malman (Carnegie Mellon University), John McDermott (Digital Equipment Corp.), Robert Schnelbach (Carnegie Mellon University)

Learning Plans and Information Requirements for Pilot Aiding 16-1
Keith Levi, David Perschbacher, Valerie Shalin, Honeywell Systems and Research Center

A Critical Look at KRITON: Moving from First to Second Generation 17-1
Marc Linster, Expert Systems Research Group, GMD

Knowledge Acquisition for Planners 18-1
Dirk E. Mahling, W. Bruce Croft, University of Massachusetts

Understanding Subtasks from a Piecemeal Collection of Knowledge 19-1
Sandra L. Marcus, Boeing Advanced Technology Center

Using Three Minimally Biasing Elicitation Techniques for Knowledge Acquisition 20-1
Mary A. Meyer, S. M. Mniszewski, A. T. Peaslee, Jr., Los Alamos National Laboratory

A Methodology and Tool for Knowledge Acquisition in KEATS-2 21-1
Enrico Motta, Tim Rajan, Marc Eisenstadt, The Open University

Developing a Tool for Knowledge Integration: Initial Results 22-1
Kenneth Murray, Bruce W. Porter, University of Texas at Austin

An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools 23-1
Mark A. Musen, Stanford University

Acquisition of Very Large Knowledge Bases: Methodology, Tools and Applications 24-1
S. Nirenburg, Ira Monarch, T. Kaufmann, I. Nirenburg, Jamie Carbonell, Carnegie Mellon University

Integration of Acquisition and Performance Systems 25-1
Alain Rappaport (Neuron Data), Brian Gaines (University of Calgary)

The Meaning Triangle as a Tool for the Acquisition of Abstract, Conceptual Knowledge 26-1
Stephen Regoczei (Trent University), Graeme Hirst (University of Toronto)

Widening the Knowledge Acquisition Bottleneck 27-1
Celine Rouveirol (Universite' Paris-Sud), Michel Manago (Intellisoft)

House Calls: Building and Maintaining a Diagnostic Rule-Base 28-1
Kalev Ruberg, Steven M. Cornick, Kay A. James, National Research Council of Canada

Using Personal Construct Techniques for Collaborative Evaluation 29-1
Douglas C. Schuler, Peter J. Russo, John H. Boose, Jeffrey M. Bradshaw, Boeing Advanced Technology Center

A Methodology for Recognizing Conflict, Correspondance, Consensus and Contrast in a Knowledge Acquisition System 30-1
Mildred L. G. Shaw, Brian R. Gaines, University of Calgary

Recent Progress in AQUINAS: A Knowledge Acquisition Workbench 31-1
David B. Shema, John H. Boose, Jeffery M. Bradshaw, Boeing Advanced Technology Center

Learning Redesign Knowledge 32-1
Evangelos Simoudis, Digital Equipment Corporation

Extracting Lexical Knowledge from Dictionary Text 33-1
Brian M. Slator, North Dakota State University

Distributed Causal Reasoning for Knowledge Acquisition: A Functional Approach to Device Understanding 34-1
Jon Sticklen (Michigan State University), B. Chandrasekaran (Ohio State University), W. E. Bond (McDonnell Douglas)

Semi-Automatic Acquisition of Conceptual Structure from Technical Texts 35-1
Stan Szpakowicz, University of Ottawa

Performance-Based Knowledge Acquisition: A Structural Foundation for Traditional Introspective Knowledge Acquisition 36-1
Deborah Tranowski, Tod Levitt, Kaitlin Riley, Advanced Decision Systems

Knowledge Acquisition at the Front-End: Defining the Domain 37-1
Brian Woodward, University of Calgary

PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages 38-1
Shun-en Xie, David F. Dumaresq, Philip H. Winne, Simon Fraser University

PROGRAM
3RD AAAI-SPONSORED KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

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

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.
Catherine Kitto, Boeing Advanced Technology Center
Sandra Marcus, Boeing Advanced Technology Center
John McDermott, Digital Equipment Corporation
Ryszard Michalski, George Mason University
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. 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 and dinners will be available 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.

SCHEDULE OF EVENTS

Sunday, November 6

2:00 - 9:00 Registration
7:00 pm Reception

Monday, November 7

Overview

9:00 - 9:40 Workshop Welcome and Introduction
9:40 - 10:20 Recent Progress in Knowledge Acquisition
Brian Gaines, University of Calgary (discussion: J. Bradshaw)

10:20 - 10:40 Break

10:40 - 11:20 A Survey of Knowledge Acquisition Techniques and Tools
John Boose, Boeing Advanced Technology Center (discussion: B. Woodward)

Strategic, Procedural, Control Knowledge; Planning and Synthesis Problems

11:20 - 12:00 Knowledge Acquisition Planning
Lawrence Hunter, Yale University (discussion: A. Rappaport)

12:00 - 1:00 Lunch

1:00 - 1:40 A Method for Acquiring Strategic Knowledge
Thomas Gruber, Stanford University (discussion: W. Gale)

1:40 - 2:20 Learning Redesign Knowledge
Evangelos Simoudis, Digital Equipment Corporation (discussion: B. Porter)

2:20 - 2:40 Break

Knowledge Acquisition Tools, Methods and Techniques

2:40 - 3:20 Acquisition of Very Large Knowledge Bases: Methodology, Tools and Applications
S. Nirenburg, Carnegie Mellon University (discussion: S. Regoczei)

3:20 - 4:00 Recent Progress in Aquinas: A Knowledge Acquisition Workbench
David B. Shema, Boeing Advanced Technology Center (discussion: M. Musen)

Dinner

Tuesday, November 8

Strategic, Procedural, Control Knowledge; Planning and Synthesis Problems

9:00 - 9:40 Understanding Subtasks from a Piecemeal Collection of Knowledge
Sandra L. Marcus, Boeing Advanced Technology Center (discussion: D. Mahling)

9:40 - 10:20 Synthesizing an Effective Control Strategy from Acquired Domain Knowledge
Michael J. Freiling, Tektronix Laboratories (discussion: D. Forsythe)

10:20 - 10:40 Break

Machine Learning and Knowledge Acquisition

10:40 - 11:20 Developing a Tool for Knowledge Integration: Initial Results
Kenneth Murray, University of Texas at Austin (discussion: S. Marcus)

Poster Session

11:20 - 12:00 Poster Session Kickoff

Knowledge Acquisition Tools, Methods and Techniques

How to Do With Grids What People Say You Can't, Jeffrey M. Bradshaw, Boeing Advanced Technology Center

PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages, David F. Dumaresq, Simon Fraser University

Knowledge Elicitation in Theory and Practice: An Integrated Program of Research and Training, Diana E. Forsythe, Stanford University

Knowledge Representation for Knowledge Elicitation, N. E. Johnson, Brunel University

A Critical Look at KRITON: Moving from First to Second Generation, Marc Linster, Institut fur Angewandte Informationstechnik

Using Three Minimally Biasing Elicitation Techniques for Knowledge Acquisition, Mary A. Meyer, Los Alamos National Laboratory

Integration of Acquisition and Performance Systems, Alain Rappaport, Neuron Data

The Meaning Triangle as a Tool for the Acquisition of Abstract, Conceptual Knowledge, Stephen Regoczei, Trent University

Widening the Knowledge Acquisition Bottleneck, Celine Rouveirol, Universite' Paris-Sud

Performance-Based Knowledge Acquisition: A Structural Foundation for Traditional Introspective Knowledge Acquisition, Deborah Tranowski, Advanced Decision Systems

Knowledge Engineering at the Front-End: Defining the Domain, Brian Woodward, University of Calgary

Strategic, Procedural, Control Knowledge; Planning and Synthesis Problems

Knowledge Acquisition for Planners, Dirk E. Mahling, University of Massachusetts

Machine Learning and Knowledge Acquisition

Learning Importance of Concepts: Construction of Representative Network, Ken-ichi Handa, ElectroTechnical Laboratory

Knowledge Acquisition from Text

Semi-Automatic Acquisition of Conceptual Structure from Technical Texts, Stan Szpakowicz, University of Ottawa

Verification and Validation of Knowledge Acquisition Techniques

A Framework for the Validation of Knowledge Acquisition, Izak Benbasat, University of British Columbia

A Methodology for Validating Large Knowledge Bases, Rajiv Enand, Carnegie Group, Inc.

Case Studies: Applications of Knowledge Acquisition Tools and Techniques

Learning Plans and Information Requirements for Pilot Aiding, Keith Levi, Honeywell Systems and Research Center

House Calls: Building and Maintaining a Diagnostic Rule-Base, Kalev Ruberg, National Research Council of Canada

12:00 - 1:00 Lunch

Knowledge Acquisition Tools, Methods and Techniques

1:00 - 1:40 Building Expert Systems with KNACK
Georg Klinker, Digital Equipment Corp. (discussion: C. Kitto)

1:40 - 2:20 Progress in Automated Knowledge Acquisition Tools: How Close Are We to Replacing the Knowledge Engineer?
Catherine M. Kitto, Boeing Advanced Technology Center (discussion: G. Klinker)

2:20 - 2:40 Break

Poster Session

2:40 - 3:20 Poster Session II

Dinner

Wednesday, November 9

Knowledge Acquisition Tools, Methods and Techniques

9:00 - 9:40 A Methodology and Tool for Knowledge Acquisition in KEATS-2
Enrico Motta, The Open University (discussion: M. Linster)

9:40 - 10:20 An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools
Mark A. Musen, Stanford University (discussion: T. Gruber)

10:20 - 10:40 Break

Case Studies: Applications of Knowledge Acquisition Tools and Techniques

10:40 - 11:20 Automated Knowledge Acquisition for a Computer Hardware Synthesis System
William P. Birmingham, University of Michigan (discussion: J. Sticklen)

11:20 - 12:00 Using Personal Construct Techniques for Collaborative Evaluation
Douglas C. Schuler, Boeing Advanced Technology Center (discussion: M. Shaw)

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

Thursday, November 10

Knowledge Acquisition from Text

9:00 - 9:40 Extracting Lexical Knowledge from Dictionary Text
Brian M. Slator, Digital Equipment Corporation (discussion: S. Szpakowicz)

Knowledge Acquisition Tools, Methods and Techniques

9:40 - 10:20 A Methodology for Recognizing Conflict, Correspondance, Consensus and Contrast in a Knowledge Acquisition System
Mildred L. G. Shaw, University of Calgary (discussion: I. Benbaset)

10:20 - 10:40 Break

10:40 - 11:20 Distributed Causal Reasoning for Knowledge Acquisition: A Functional Approach to Device Understanding
Jon Sticklen, Michagan State University (discussion: W. Birmingham)

11:20 - 12:00 An Interactive Performance Based Knowledge Acquisition Methodology
William A. Gale, AT&T Bell Laboratories (discussion: N. Johnson)

12:00 - 1:00 Lunch

Working Groups

Afternoon Integration of ML and KA (Reporting: T. Gruber, B. Porter)
KA Tools (Reporting: M. Musen, B. Woodward)
Rep Grid Methods for KA (Reporting: J. Bradshaw, M. Shaw)

Synthesis Tasks and KA: Planning, Scheduling, Configuration, Design... (Reporting: M. Freiling, S. Marcus)
Text Analysis (Reporting: B. Slator, S. Szpakowicz)
Validation and Verification of KA (Reporting: B. Gale, N. Johnson)

2:20 - 2:40 Break

Banquet Dinner with Annual Bear Tossing Speech

Friday, November 11

Working Group Presentations

8:30 - 9:00 Integration of ML and KA (Reporting: T. Gruber, B. Porter)
9:00 - 9:30 KA Tools (Reporting: M. Musen, B. Woodward)
9:30 - 10:00 Rep Grid Methods for KA (Reporting: J. Bradshaw, M. Shaw)

10:00 - 10:15 Break

10:15 - 10:45 Synthesis Tasks and KA: Planning, Scheduling, Configuration, Design... (Reporting: M. Freiling, S. Marcus)
10:45 - 11:15 Text Analysis (Reporting: B. Slator, S. Szpakowicz)
11:15 - 11:45 Validation and Verification of KA (Reporting: B. Gale, N. Johnson)

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

12:15 - 1:00 Farewell Lunch and Bear Souvenirs

LIST OF PARTICIPANTS AND AUTHORS

3RD AAAI-SPONSORED KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff, Canada, November 6-11, 1988

Co-Chairs: **John H. Boose** and **Brian R. Gaines**

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

William P. Birmingham
University of Michigan, Department of Electrical Engineering and Computer Science, Ann Arbor, Michigan 48109, USA

W. E. Bond
McDonnell Douglas, Research Lab, St. Louis, MO 63166, USA

John H. Boose
Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124 USA

Casey Boyd
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Jeffrey M. Bradshaw
7L-64, Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

Bruce G. Buchanan
Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 701 Welch Road, Building C, Palo Alto, CA 94304, USA

Jamie Carbonell
Center for Machine Translation, Carnegie Mellon University, Pittsburgh, PA 15213, USA

B. Chandrasekaran
LAIR - CIS Dept., 2036 Neil Ave. Mall, Ohio State University, Columbus, OH 43210, USA

William Clancey
Institute for Research on Learning, 3333 Coyote Hill Rd., Palo Alto, CA 94304, USA

Stanley P. Covington
7L-64, Knowledge Systems Laboratory
Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

Steven M. Cornick
Institute for Research in Construction, National Research Council of Canada, Ottawa, Ontario, Canada K1A 0R6

W. Bruce Croft
Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, USA

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

David Dong
Digital Equipment Corp., DLB5-3/E2, 290 Donal Lynch Blvd., Marlboro, Massachusetts 01752, USA

David F. Dumaresq
Instructional Psychology Research Group, Faculty of Education, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6

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

Rajiv Enand
Carnegie Group, Inc., 5 PPG Place, Pittsburgh, PA 15222, USA

Diana E. Forsythe
Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 701 Welch Road, Building C, Palo Alto, CA 94304, USA

Michael J. Freiling
Tektronix Laboratories, MS 50-662, PO Box 500, Beaverton, OR 97077, USA

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

William A. Gale
AT&T Bell Laboratories, Room 2C278, 600 Mountain Avenue, Murray Hill, NJ 07974, USA

Thomas Gruber
Knowledge Systems Laboratory, Department of Computer Science, Stanford University, 701 Welch Road, Building C, Palo ALto, CA 94304

Ken-ichi Handa
Machine Inference Section, ElectroTechnical Laboratory, Umezono 1-1-4, Tsukuba City, JAPAN, 305

Graeme Hirst
Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4

Lawrence Hunter
Department of Computer Science, Yale University, New Haven, CT 06520, USA

Shun Ishizaki
Machine Inference Section, ElectroTechnical Laboratory, Umezono 1-1-4, Tsukuba City, JAPAN, 305

Kay A. James
Institute for Research in Construction, National Research Council of Canada, Ottawa, Ontario, Canada K1A 0R6

N. E. Johnson
Department of Computer Science, Brunel University, Uxbridge, Middlesex UB8 3PH, U.K.

Gary S. Kahn
Carnegie Group, Inc., 5 PPG Place, Pittsburgh, PA 15222, USA

T. Kaufmann
Center for Machine Translation, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Catherine M. Kitto
7L-64, Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

Georg Klinker
Digital Equipment Corp., DLB5-3/E2, 290 Donal Lynch Blvd., Marlboro, Massachusetts 01752, USA

Keith Levi
Honeywell Systems and Research Center, Honeywell, Inc., 3660 Technology Dr., Minneapolis, MN 55418, USA

Tod Levitt
Advanced Decision Systems, 1500 Plymouth St., Mountain View, CA 94043, USA

Marc Linster
Forschungsgruppe Expertensysteme, Institut fur Angewandte Informationstechnik, GMD, Postfach 1240, Sankt Augustin 1, FRG

Dirk E. Mahling
Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, USA

Jon Malman
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Michel Manago
Intellisoft, 28, Av Georges Clemenceau, 91400 Orsay, France

Sandra L. Marcus
7L-64, Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

John McDermott
Digital Equipment Corp., DLB5-3/E2, 290 Donal Lynch Blvd., Marlboro, Massachusetts 01752, USA

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

Ryszard Michalski
Center for AI, Science and Technology Building, Room 301, George Mason University, 4400 University Dr., Fairfaz, VA 22030, USA

Robert A. Mills
Ford Motor Company, Dearborn, MI, USA

S. M. Mniszewski
MS F600, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Hiroshi Motoda
Advanced Research Laboratory, Hitachi, Ltd., Kokubunji, Tokyo 185, Japan

A. T. Peaslee, Jr.
MS F600, Los Alamos National Laboratory, Los Alamos, NM 87545, USA

Ira A. Monarch
Center for Machine Translation, Carnegie Mellon University, Pittsburgh, PA 15213, USA

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

Kenneth Murray
Department of Computer Science, AI Laboratory, Taylor Hall 2.124, University of Texas at Austin, Austin, TX 78712, USA

Mark A. Musen
Medical Computer Science Group, Knowledge Systems Laboratory, Stanford University, Stanford, CA 94305, USA

I. Nirenburg
Center for Machine Translation, Carnegie Mellon University, Pittsburgh, PA 15213, USA

S. Nirenburg
Center for Machine Translation, Carnegie Mellon University, Pittsburgh, PA 15213, USA

David Perschbacher
Honeywell Systems and Research Center, Honeywell, Inc., 3660 Technology Dr., Minneapolis, MN 55418, USA

Bruce W. Porter
Department of Computer Science, AI Laboratory, Taylor Hall 2.124, University of Texas at Austin, Austin, TX 78712, USA

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

Alain Rappaport
Neuron Data, 444 High St, Palo Alto, CA 94301, USA

Stephen Regoczei
Computer Studies, Trent University, Peterborough, Ontario, Canada K9J 7B8

Kaitlin Riley
Advanced Decision Systems, 1500 Plymouth St., Mountain View, CA 94043, USA

Celine Rouveirol
Laboratoire de Recherche en Informatique, U.A. 410 du CNRS, Bat 490, Universite' Paris-Sud, 91405 Orsay Cedex, France

Kalev Ruberg
Institute for Research in Construction, National Research Council of Canada, Ottawa, Ontario, Canada K1A 0R6

Peter J. Russo
7L-64, Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

Robert Schnelbach
Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Douglas C. Schuler
7L-64, Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

Valerie Shalin
Honeywell Systems and Research Center, Honeywell, Inc., 3660 Technology Dr., Minneapolis, MN 55418, USA

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

David B. Shema
7L-64, Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services, P.O. Box 24346, Seattle, WA 98124, USA

Daniel P. Siewiorek
Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Evangelos Simoudis
AI Applications Group, Digital Equipment Corporation, 290 Donald Lynch Blvd., Marlboro, MA 01752, USA

Brian M. Slator
Department of Computer Science, North Dakota State University, 300 Minard Hall, SU Station, P.O. Box 5075, Fargo, ND 58105, USA

Jon Sticklen

AI/KBS Group, Computer Science Department, A714 Wells Hall, Michigan State University, East Lansing, MI 48824, USA

Stan Szpakowicz
Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada K1N 6N5

Deborah Tranowski
Advanced Decision Systems, 1500 Plymouth St., Mountain View, CA 94043, USA

C. M. Tomlinson
Department of Computer Science, Brunel University, Uxbridge, Middlesex UB8 3PH, U.K.

Philip H. Winne
Instructional Psychology Research Group, Faculty of Education, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6

Brian Woodward
Knowledge Sciences Institute, Department of Computer Science, University of Calgary, 1500 University Dr. NW, Calgary, Alberta, Canada T2N 1N4

Shun-en Xie
Instructional Psychology Research Group, Faculty of Education, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

A FRAMEWORK FOR THE VALIDATION OF KNOWLEDGE ACQUISITION

Izak Benbasat and Jasbir S. Dhaliwal
Faculty of Commerce and Business Administration
University of British Columbia
Vancouver, B.C., Canada
V6T 1Y8
EMail: USERIZAK@UBCMTSG

ABSTRACT
Current knowledge acquisition literature provides few guidelines as to the appropriate set of validation criteria and techniques to be applied at various stages of the knowledge-based systems (KBS) development process. Utilizing a representational homomorphism definition of validation, this paper proposes a framework where validation evolves as a sequence of stages paralleling the different stages of the KBS development life cycle. This framework incorporates an inventory of validation methods and identifies the entities to be measured, the types of evidence to be collected, the criteria to be applied and the type of comparisons to be made to assess validity.

INTRODUCTION
With the recent proliferation of knowledge-based systems (KBS) and the attention paid to their development, an increasing number of authors have expressed interest in methods for the validation of these systems (O'Leary, 1987; Green and Keyes, 1987; Shaw and Woodward, 1988). It can be argued that the need for validation is greater for KBS systems than for conventional computer systems. Sell (1985) notes that while the processing of traditional computer systems rests on established or well-entrenched scientific, professional, or business laws and practices, knowledge systems are based on heuristic rules-of-thumb which have never been subjected to rigorous examination. In addition, KBS go beyond providing decision support; they have the potential to substitute human judgmental decision-making performance in a variety of critical tasks such as process control, design, diagnosis and planning. These concerns heighten the need and importance of validation in the KBS development process.

According to O'Keefe et al. (1987) validation of KBS has mostly been handled in an *ad hoc*, informal way. In addition to the concern about properly testing systems as to their value and correctness, there is the need to be able to compare and evaluate the multitude of knowledge acquisition (KA) methods and tools (both manual and automated). These have been suggested and developed in the last decade in response to calls for research into overcoming the KA bottleneck. Now that this need has to some extent been filled, KA researchers are turning their attention to validation issues.

A number of validation techniques have been suggested in the literature and/or have been used in practice. As well, there is a wealth of information dealing with issues of validity in the fields of simulation modelling, development of computer software, and accounting (auditing). *However few guidelines exist as to the appropriate set of validation techniques and criteria to be applied at various stages of the KBS development process.* This paper discusses an inventory of methods for the validation of the KA process in the development of KBS. While the emphasis is on the validation of KA, some of the methods to be discussed are also applicable to determining the

validity of KBS since the validity of construction directly influences the validity of performance.

The paper begins by providing definitions for the different issues to be covered. A difficulty of attempting to integrate the literature is the variety of terminology used and the different meanings given to the same terms by various authors. A life cycle model for KBS development is then presented in order to place the types of validation methods to be suggested within the context of the different phases of this cycle. Next, major categories of validity evidence are introduced as general classes of measurement concepts. This is followed by a discussion of a framework of tests and techniques for validating the KA process.

DEFINITIONS

There are various definitions of validity used within the context of computer programming, KBS, and simulation modelling. These are discussed in this section and contrasted to other terms such as reliability, verification, and evaluation with which validity is sometimes used interchangeably. The elaboration of different types of validity and the techniques to measure them will be covered in subsequent sections.

In psychometrics validity is the degree to which a measuring instrument actually does what it is intended to do (Nunnally, 1967, p. 65). If the objective of the KA process is to capture the knowledge of an expertise source then *the validity of the KA process is the degree of homomorphism between the representation system, i.e., KBS, and the system that it is supposed to represent, i.e., the expertise source* (Vandierendock, 1975).

This *representational homomorphism* definition of validity is consistent with Bobrow's (1975) notion that at an abstract level a representation can be viewed as "the result of a selective mapping of aspects of the world." A world-state (expert source) can be modelled into a knowledge-state (KBS) and questions about the world-state can be answered either by observing the world-state or by questioning the corresponding knowledge-state. Additionally, just as the world-state can be changed by an action (e.g., experience with a new situation), its mapped knowledge-state counterpart can be changed by a model operation, with the resulting knowledge-state being an exact representation of the changed world-state.

Furthermore, the fact that validation involves comparing a representation to its *source* world-state to determine the degree of homomorphism suggests that validation criteria should include: domain and range correspondence (representational adequacy), operational (event) correspondence, grain size (level of detail) correspondence, abstraction level correspondence, meta-knowledge correspondence, construct correspondence, and correspondence in terms of the set of primitives used to provide expressive power. If the source is a human expert, validation also involves assessing how natural and psychologically valid the representation is as compared to the source. It is also important at this juncture to note that absolute KA validity is infeasible as the KBS being developed is but a slice/model of reality. However, it is critical that validation tests be undertaken to ascertain an adequate level of validity.

The definition above differs from that suggested for software validation by Adrion et al. (1982, p. 188): "validation is the determination of the correctness of the final program from a development project with respect to the user needs and requirements." What is discussed in this paper is not validation against a set of project specifications but validation against the knowledge of the expert sources.

Verification is associated with, but not the same as, validation. According to Adrion et al. (1982), *verification is the demonstration of consistency, completeness and correctness of the software at each stage and between each stage of the software development life cycle.* Verification is performed with respect to a set of specifications from a prior stage of the development process. Oren (1986) defines verification as the evaluation of a computerized model with respect to its conceptual model. Assuming that one starts with a valid specification of expertise, the verification process increases the chances that the software created will be valid.

A *reliable instrument is one that contains a low degree of measurement error.* Reliability of a KA method is measured by the degree of accuracy and precision of the method. A KA tool that generates the same knowledge-base from the same expert source over repeated applications is a reliable one. However, high reliability while being a necessary condition for validity is not a sufficient condition. For example, a KA tool that consistently acquires only a given subset of the expert's knowledge but not all of his appropriate knowledge in a specified domain is reliable but not completely valid.

Evaluation is different from validation. According to Gaschnig et al. (1983), there are two aspects to evaluation: evaluation by experts and evaluation by users. *Evaluation by experts is to determine both the quality of the knowledge-base and of the advice that the KBS provides.* Note that while evaluation by expert(s), especially the one(s) after which the system has been modelled, will be one of the techniques recommended for validation, evaluation by the independent experts is not necessarily the same as validating the KA process. That is, the KA process could fully capture the knowledge of a source expert (i.e., have high representational validity), while at the same time what has been captured, i.e., the expertise, could fall short of a given quality level as defined by another set of experts (e.g., see Yu et al., 1979). If this distinction is not made then as Shaw and Woodward (1988) argue, it becomes impossible to separate the KA tool or method from the quality of the expert.

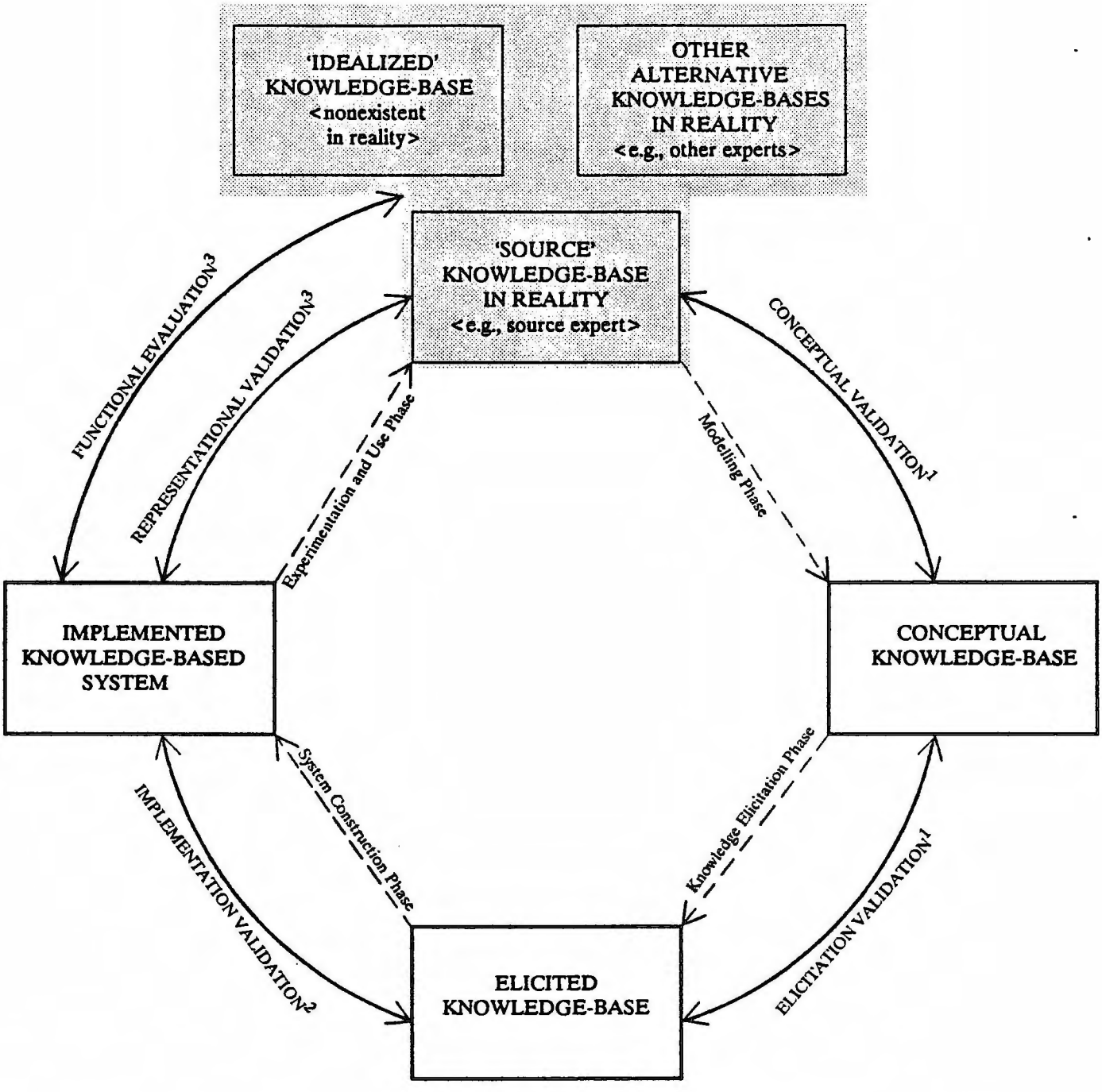
Evaluation by users is associated with both the usefulness and the usability of the KBS. Usefulness refers to the utility and benefits derived from the system; usability is associated with the understandability of results, human-engineering aspects, ease of modifications, etc.

To summarize, there are several of types of evaluation, as distinct from validity, that will be referred to in this paper:

(1) Evaluation of the quality of KBS: quality as determined by a group of experts in the context of the problem to which the KBS is applied to or measured objectively based on quantitative criteria of performance.
(2) Evaluation of the utility of KBS: utility as identified by users based on actual or perceived benefits as applied to their problems.
(3) Evaluation of the usability of KBS: ease of use and acceptance by the user community.

KNOWLEDGE-BASED SYSTEM DEVELOPMENT LIFE CYCLE

Substantiation of the validity of a knowledge-base, besides being an essential part of the KBS development process, is itself a process. It involves undertaking tests and evaluations *within the different stages* of the development process. It is insufficient to validate a final implemented KBS in terms of performance. *Rather validation should unfold as a sequence of stages paralleling the different stages of the KBS development life-cycle.*



Notes: [1]Conceptual and elicitation validation constitute **Knowledge-Acquisition Validation.**

[2]Implementation validation can be termed **Verification.**

[3]Both functional evaluation and representational validation constitute **Knowledge-Based System Validation.**

FIGURE 1: A VALIDATION-BASED LIFE CYCLE MODEL OF THE KNOWLEDGE-BASED SYSTEM DEVELOPMENT PROCESS

Figure 1 depicts a life cycle model of the KBS development process. It differs from other similar models (e.g., Hayes-Roth et al., 1983, and Fellers, 1987) in that it specifically isolates, at different stages of the life cycle, three progressively more developed knowledge-bases, all of which require different combinations of validation criteria, measurements and techniques. The model comprises four sequential activity-phases -- *modelling, knowledge elicitation, system construction,* and *experimentation and use* -- each separated by one of the three knowledge-base (KB) states. This section will present an overview of the validation aspects of this life cycle as a prelude to a more detailed discussion of the validation tests and criteria required for assessing each of the three knowledge-bases.

The conceptual knowledge-base represents an abstraction of the source knowledge-base in the real world. Hayes-Roth et al. (1983) describe this initial phase as the isolation of the knowledge relevant to solving the problem and identifying the key elements of the problem description. The outcome of this process can be expressed in the form of words, diagrams, schematics, or by mathematical representations (Oren et al., 1985). *Conceptual validation* attests to the quality of the modelling process by which conceptual knowledge-base specifications are obtained from the expert source. It is important here to ensure that the assumptions and theories underlying the conceptual knowledge-base are correct and that it is a representation of reality that is reasonable for the intended application.

Elicitation validation provides assurance regarding the completeness and correctness of the *KA process* by which the conceptual knowledge-base is instantiated and extended into an elicited knowledge-base. Together, conceptual validation and elicitation validation constitute what is termed in the literature as validating the KA process. Furthermore, we view the *validation of KA tools as being a subset of elicitation validation.*

Implementation validation is aimed at ensuring the quality of the translation/-implementation process that transforms the elicited knowledge-base into a functional KBS (i.e., a working system or prototype). It is analogous to *verification* in traditional software engineering (see Definitions section). The implemented knowledge-base can be expected to have equal but not greater validity than the elicited knowledge-base. This is because implementation validation is limited to verifying the system construction process that translates a set of specifications (including a knowledge specification) into a fully functional KBS.

Subsequent to implementation validation, the functional KBS can further be assessed in two ways. In our model these are termed functional evaluation and representational validation, and together they constitute what is often referred to as KBS validation. *Functional evaluation* resembles the common interpretation of KBS evaluation and refers to performance evaluations of quality, utility, and usability as described in the Definitions section.

Representational validation attests to the match between the characteristics of the implemented KBS and those of the expert source. Following our earlier definitions of representation and validity, this validation takes two forms: 1) *structural matching* which refers to the validation of knowledge representation, and processing (reasoning) logic, and 2) *behavioral matching* which refers to the similarity between the output or performance of the KBS and that of the source expert. While such validity can strictly be obtained only from comparisons to the expert source that the KBS is attempting to represent, other comparisons could be made to gather additional evidence. This relates to Feigenbaum and Feldman's (1963, pp. 274) assertion, (made in the

context of modelling human thought processes), that "the actual comparison or test procedure depends on whether the model represents a particular individual or a generalized or idealized individual." These include comparisons with alternative nonsource expert knowledge-bases existing in reality (usually in a less developed or dispersed/nonintegrated form), and idealized knowledge-bases framed in the minds of the system developers or users. An example of the former is the comparisons made to specialist and nonspecialist physicians in the validation of *MYCIN* (Buchanan and Shortliffe, 1985). The latter is exemplified by Vedder and Mason's (1987) development of an idealized KBS for aiding decision makers in hostage-taking incidents, based on what Johnson (1983) terms reconstructed sources of expertise such as normative theories, real case descriptions, and simulations of hostage-taking situations.

It is important to note that while the life cycle is presented in a somewhat static sequential manner in the interest of clarity of presentation, in reality the different phases overlap significantly, and there are multiple iterations of each phase. This is especially so in the case of rapid prototyping using an automated KA tool (Gaines, 1987). The popularity among knowledge engineers of this methodology in developing KBS is grounded in two factors. First, unlike traditional databases (which capture, store, and offer for use observable, verifiable facts about the real-world), knowledge-bases attempt to model and replicate human conceptual knowledge which is not directly observable in nature. The second factor is our poor understanding of the cognitive aspects of human utilization of such knowledge and the undeveloped state of methodologies for accessing it. The multiple life cycle iterations of prototyping suggest that it can be viewed as a *validation oriented* systems development methodology, for it requires repeated validation and evaluation to provide justification and data for further development.

However, this focus on rapid prototyping has had two major outcomes. First, the early need of a functional prototype and the subsequent repeated iteration (and, thereby, integration) of KA and implementation makes it likely that there will be a considerable *representational mismatch* (Hayes-Roth et al., 1983) between the source expert's knowledge-base and its machine-implemented version. This is because decisions about representation schemes and inference engines have to be made early and KA takes on a definite implementational orientation. As a consequence, validating the KA process (as distinct from KBS validation) including the KA tools used, has taken on increased importance as a critical element of the KBS development process. Second, the early commitment to implementation and the constant user involvement tend to emphasize implementation and functional evaluation at the expense of conceptual, elicitation, and representational validity. Thus, while prototyping could provide functional and usable KBS, it affords little guarantee as to whether they are truly valid.

TYPES OF VALIDITY EVIDENCE

Validation is the process by which a series of assessment tests are undertaken for the purposes of accumulating a web of evidence pertaining to the validity of a KBS (as defined earlier in the Definitions section). There are many validity assessment tests and methods which have been suggested in the KBS, KA and simulation literature. To facilitate their discussion in later sections of this paper, this section describes in the context of KBS and KA validation the three major types of validity evidence currently utilized in the psychological testing literature (American Psychological Association, 1985).

Content-related evidence is collected to assess the adequacy of the process with which the specified domain has been sampled and to determine the adequacy of the content generated. Johnson et al. (1986) present a procedure that attempts to validate

expertise by modelling the operative knowledge of a task. They define *operative knowledge* as being that knowledge which is necessary for accomplishing a task or class of tasks. This is an example of validity obtained by gathering content-related evidence. This evidence demonstrates the extent to which the knowledge represented in a KBS is representative of some defined universe or domain of content. It is advisable to validate content prior to conducting predictive assessment tests. Predictive tests are costly and time consuming since they usually require a working system and the use of experts to make comparisons.

The process of gathering content-related evidence includes the definition of the domain, surveying the domain, elicitation of the knowledge-base (KB) facts and rules, demonstrating that the content elicited is representative of the domain, and the development of criterion to exclude items that are not part of that domain.

The adequacy of process is evaluated by having an independent party assess the extent to which the steps described above are followed properly. The adequacy of content can be evaluated by methods such as a "structured walkthrough" which essentially involves bringing together a group of people, e.g. experts, knowledge engineers, and users, to examine the contents of a representation of the KB at different stages of the KBS life cycle. The intention is to find errors, inaccuracies, omissions, contradictions, and redundancies. An example of this is Shaw and Woodward's (1988) use of source experts to examine the contents of KB elicited from them and the use of another group of experts to detect omissions.

Content-related evidence is more important for assessing conceptual and elicitation validation (KA validation), that is, prior to the development of a working prototype KBS in which case criterion-related evidence could be collected. There are several concerns associated with collecting content-related evidence (O'Leary, 1987) namely,

(a) the issue of finding a representation that adequately represents the contents and is also understandable to the parties examining the content,
(b) the cost and time of people involved in the process,
(c) the expertise and reliability of people conducting the assessment,
(d) the limitations of human information processing; the substantial cognitive resources involved in examining the contents of a complex KB and in understanding the interaction between its parts.

Criterion-related evidence refers to he assessment of a construct by demonstrating a relationship between its behavior and some other criterion measure. This is the most commonly used method in the area of KBS validation. An example is comparing the performance of the KBS to that of the expert after which it is modelled.

Such evidence can be collected in three ways: predictive, concurrent, and post-dictive. Predictive refers to the situation where criterion data is collected at some time in the future. For example, the product specifications designed by a KBS such as R1/XCON (Bachant and McDermott, 1984) could be compared to customer complaints about missing parts identified at a later date. Turing tests might involve the concurrent collection of criterion data from the KBS and the human expert. Post-dictive refers to the case where the criterion data has been collected in the past, such as comparing KBS output against test-cases with known outcomes.

Criterion data could be collected from the source expert, other nonsource experts, both realistic and hypothetical test-cases, and from the predictions of theory. It is important to note that the way criterion data is collected might often be a source of error. A discrepancy between the construct (e.g., KBS) and the criterion-evidence

might be due to a real mismatch or incorrect measurement of either the criterion or the construct, or both. Therefore in the validation framework to be presented in the next section (and summarized in Tables 1, 2 and 3) this issue is considered under data collection and experimentation *method* assessment.

A final point on criterion-related assessment is that in addition to output behavior, the structure or process by which the output is generated could also be evaluated. Examples of such comparison criteria might be the number of attributes and/or alternatives considered, the number of iterations the system went through, the breadth or depth of search correspondence, and event or operational correspondence.

Construct-related evidence is collected to assess whether a measure is an appropriate operational definition of the construct it purports to represent. Constructs are variables that are abstract and not directly observed, for example, intelligence, and anxiety. Therefore, in such cases, criterion-related tests are not applicable since there is no direct or better measure of such criterion. This type of evidence is especially important in KA validation when the comparisons are made to idealized, composite or reconstructed sources of expertise as was discussed in the previous section, in which case the criterion is difficult to measure directly.

Construct-related evidence is not always easy to obtain and specify. It is collected by first developing an operational definition of the construct (in this case a KB or KBS) and then developing hypotheses about how this measure relates to other measurable variables. In addition, a conceptual framework should be established to show how the construct relates to and is distinguished from other constructs and measures associated with it. Convergent and divergent evidence, as specified by the conceptual framework established, could then be sought. Shaw and Woodward (1988) provide examples of such evidence. They mention a category labelled *objective validity* which measures the degree to which the facts in the KB agree with well established facts in the domain. This is an example of convergent evidence. On the other hand, conflict with the predictions of a theory would constitute divergent evidence. Comparing the KBs generated using the same KA method by two or more individuals with similar levels of expertise would be helpful for gathering convergent and divergent evidence as to the validity of a KA tool.

DISCUSSION OF VALIDITY ASSESSMENT TECHNIQUES

The KBS development life cycle (Figure 1) comprises three progressively more developed knowledge-bases. Tables 1, 2 and 3 describe the types of assessment tests necessary for validating and evaluating these knowledge-bases representing the different stages of KBS development. It is adopted from a framework proposed by Oren et al. (1985) for quality assurance in the simulation modelling area.

Conceptual Knowledge-Base Assessment

The conceptual modelling phase produces two kinds of output both of which need to be assessed: a specification of the conceptual knowledge-base that is independent of any internal representation or control process formalism and a battery of test-cases (see Table 1).

Assessment of:	With Respect To:	Type of Test	Definition	Comment
Conceptual Knowledge-Base	Problem Being Solved	Conceptual KB Qualification	Evaluation of a KB with respect to the problem being solved (KB relevance and adequacy).	Is it the right KB for solving the problem and is it adequate for that purpose?
	Source Expert	Conceptual KB Validity (Realism)	Validation of the KB with respect to the expert source being modelled	How faithful of a representation is it?
	Modelling Formalism	Formal Checking	Evaluation of conceptual KB with respect to modelling formalism.	Are conceptual KB specifications correct and complete in terms of the modelling formalism?
Test-Cases	Problem Domain	Test-Case Applicability	Evaluation of test-cases with respect to problem being solved.	Are the test-cases adequate to cover problem domain?
	Source Expert	Test-Case Validity	Validation of test-cases with respect to the expert source.	Are the test-cases relevant to and do they tap the expertise being modelled?

TABLE 1: CONCEPTUAL KNOWLEDGE-BASE ASSESSMENT

Assessment of Conceptual Specifications - Three types of assessments can be performed on the conceptual knowledge-base specifications: 1) **Knowledge-base qualification** to determine its relevance and adequacy in terms of the objectives of the intended application or problem being solved; 2) **Knowledge-base validity** to determine its structural faithfulness to the source expert; and 3) **Formal checking** to determine correctness and completeness in the application of the modelling formalism selected.

A suggested technique for knowledge-base qualification is the use of specification walkthroughs involving both source and nonsource experts, knowledge engineers, and prospective users of the KBS. Prospective user involvement ensures that the correct system is specified for the problem to be solved. The purpose here is to examine modelling documentation such as decision trees and knowledge hierarchy charts to isolate problems, inaccuracies, ambiguities, and omissions in the conceptual KB. Knowledge-base validity (realism) invokes comparisons between the specifications and the real source of expertise based on a set of structural criteria. These are similar to structural assessment criteria for the elicited KB and are discussed below. Other

secondary techniques that are applicable to assessing the conceptual KB include assumptions challenging and seeking convergent and divergent validity from related theories and practical observations in the application domain.

Assessment of Test-Cases - The test-cases serve three purposes: as a basis for problem definition at the conceptual stage, as examples for source experts and knowledge engineers to work through at the elicitation stage, as well as a basis for output sensitivity analysis under different scenarios at both the elicitation and implementation stages of the life cycle. They need to be assessed with respect to the problem domain (**test-case applicability**) and to the source of expertise being modelled (**test-case validity**). This is important as nonrepresentative and unrealistic test-cases can lead to incomplete and erroneous KA at a later stage. Quinlan (1987) points out that unrealistic test-cases can cause experts to reason at overly and uncharacteristically abstract levels and thereby hinder detailed KA.

Elicited Knowledge-Base Assessment

Starting with the conceptual specifications, the KA process produces an *elicited knowledge-base*. This takes the form of a 'complete' expertise specification formalized to some selected knowledge representation (for example, rules, frames, or semantic nets) but in *nonmachine-readable* form. It can be assessed in terms of its match with its prior conceptual specifications, representational structure, input-output behavior and sensitivity, and the elicitation formalism used (see Table 2).

Such assessments necessitate making comparisons of the elicited KB to both the source expert being modelled and other existing expertise sources. Therefore, an examination of the data collection and experimentation methods required to make such comparisons (**data collection veracity**) is also warranted to account for the validity and reliability of the data collected (for example, variances arising from expert fatigue, motivation, and communication problems in test data collection). (Note that 'Method' is included as part of the assessment tests in both Table 2 and 3 for elicited and implemented KB, respectively.)

Assessment of Structure - Assessments of structure have two aspects: comparisons to the source expert (**structural validity**) and to other expertise sources (**structural comparisons**). While the former measures the representational homomorphism of the elicited knowledge-base by collecting content-related evidence, the latter largely provides construct-related evidence. Specific assessment techniques include comparing the objects and relations in the elicited KB to those of both source and nonsource experts. Shaw and Woodward (1988) provide an example of this by having the source experts rate their own entailments as generated by the *KSS0* system (structural validity), as well as rating the entities and attributes generated by other experts (structural comparison). This attests to what in the Definitions section was termed *domain and construct correspondence*. Another technique is to compare the number of values an attribute or property can take in the elicited model as compared to the source expert. This is a test for *range correspondence* and attests to both the accuracy and exhaustiveness of variable ranges. Additionally, procedural traces obtained from the source expert using protocol analysis (Todd and Benbasat, 1987) or structured walkthroughs (Yourdon, 1986) could be compared to the processing steps in the elicited KB to determine the *operational or event correspondence* between the two structures. This will reveal differences, if any, in processing strategies between the two. For example, while a mathematical model that predicts an individual's output based on input cues processed may reliably lead to an outcome consistent with that of the individual, protocol analysis may reveal that the underlying process (structure) by which the individual arrives at the decision is radically different.

Assessment of:		With Respect To:	Type of Test	Definition	Comment
Elicited Knowledge-Base	STRUCTURE	Source Expert	Structural Validity	Validation of the structure with respect to the *perceived* structure of source expert.	Structural Homomorphism
		Other Source Expert	Structural Comparison	Evaluation of the structure of a KB with respect to structure of some other KB, if available.	Get other experts to judge structural aspects of elicited KB.
	BEHAVIOR	Source Expert Behavior	Behavioral Validity	Validation of KB behavior (output) and expert behavior generated under same conditions.	Logical walkthrough (a weak form of validation).
		Other Expert Behavior	Behavioral Comparison	Comparison of behavior of elicited KB with behavior other KB's under similar conditions.	Compare to other experts. Inter-expert validation (not always possible).
	VERIFICATION	Conceptual KB	Elicited KB Verification	Evaluation of Elicited KB with respect to conceptual KB specification.	Is elicited KB consistent with specifications of conceptual KB?.
		Elicitation Formalism	Formal Checking	Evaluation of Elicited KB specification in terms of the elicitation formalism used (consistency and completeness checks).	e.g., equivalent of ensuring -rules are in proper format & complete -explanations are in proper format & complete.
	METHOD	Source Expert	Data Collection Veracity	Evaluation of the validity and reliability of test data collected from the source expert.	Identify instrumentation and measurement errors in test data collection.

TABLE 2: ELICITED KNOWLEDGE-BASE ASSESSMENT

Another technique for assessing the structural validity of an elicited KB follows from the need to attest to the correspondence between the *cognitive-primitives* used in the KB and by the source expert. Rappaport (1987, p. 15-4) notes that "when a rule is written, it may have been considered by the expert as a cognitive unit; however most efficient interpreters will not treat it as such, thereby reducing the structural fit." He further suggests that knowledge elicitation tools should allow a level of primitives design to fit the way human source experts express their knowledge. The availability of a set of natural primitives enhances the expressive power of a representation just as control structures determine the economy of reasoning in a model. Thus a comparison of these should be integral to any assessment of representational homomorphism.

Search-scope correspondence (e.g., the alternatives considered) and *grain-size correspondence* (e.g., the level of detail) could be assessed at this stage by having the source expert review the elicited KB and provide agreement ratings. Closely related to grain-size and cognitive primitives comparison, is the assessment of *abstraction level correspondence* and *meta-knowledge correspondence*. Both of these can only be assessed with respect to the source expert and not to other experts. This follows from studies of expertise (Chi et al, 1981) that have shown that individuals with higher levels of expertise reason at more abstract levels, and also possess and use more meta-knowledge. Such assessments can be made by examining the elicited KB (e.g., rule base) to identify the number of meta-knowledge rules being used. Also assessments can be made of the protocol traces to determine the abstractness of reasoning the source expert uses and this can be compared to the reasoning inferred from the elicited KB.

Assessment of Behavior - Assessment of the behavior of the elicited KB is less useful than the assessment of its structure as was discussed above. It is performed with respect to two criteria: source expert behavior (**behavioral validation**), and other expert behavior (**behavioral comparison**). Although the KB is not yet in an operational form, and criterion-related evidence cannot be easily collected, it is still possible to logically validate the behavior of segments of the elicited KB prior to implementation. This can be done by collecting content-related evidence through techniques such as structured logical walkthroughs by both experts and knowledge engineers. For example, missing rules can be detected by logically tracing back incorrect conclusions (output). The initially assessed test-cases should prove useful for this purpose. In addition, manual sensitivity analysis is also a valuable technique at this stage, for besides providing the knowledge engineer with assessment feedback, it also facilitates additional KA.

Elicited KB Verification - As reflected in Table 2, assessment of the elicited knowledge-base includes two verification-oriented tests: **verification** against the conceptual KB specifications and verification against the specifications of the elicitation formalism used (**formal checking**). The former serves to verify that the structuring resulting from a commitment to an internal representational formalism and the acquisition of detailed domain knowledge have not made the elicited KB inconsistent with the earlier conceptual specifications. Formal checking includes syntactic examination of the elicited KB and the use of templates for rules to ensure consistency.

Implemented Knowledge-Based System Assessment

Five categories of assessments are needed at this phase (see Table 3). The first four are: verification assessments relating to the system construction process, structural and behavioral assessments dealing with the representational validity of the KBS,

Assessment of:		With Respect To:	Type of Test	Definition	Comment
Implemented Knowledge-Base	STRUCTURE	Source Expert	Structural Validity (Realism)	Validation of KBS structure against *perceived* structure of real-system.	Determination of structural homomorphism.
		Other Experts	Structural Comparison	Evaluation of KBS structure against structure of other KB's	Is there content validity in terms of structure?
	BEHAVIOR	Source Expert	Behavioral Validity	Validation of KBS behavior with respect to source expert.	Equivalent of the Turing test.
		Other Experts	Behavioral Comparison	Evaluation of the KBS behavior against non-source experts.	How does the output measure up to other expert performance?
		Different scenarios (test-cases)	Sensitivity of KBS Behavior	Comparison of KBS behavior (output) under different scenarios.	Sensitivity Analysis of parameters
	VERIFICATION	Elicited KB	Implementation Verification	Evaluation of KBS with respect to the elicited KB specifications.	Was the translation of elicited KB correct?
		Implementation Formalism	Formal Checking	Evaluation of KBS in terms of structures & rules of the implementation formalism	Is the KBS correct & complete in terms of hardware & software?
		Software Engineering Requirements	Software Quality Assurance	Evaluation at KBS with respect to software quality standards.	Were pre-specified quality assurance requirements followed?
	USABILITY	Source Expert Utilization	Usage Comparison	Validation of KBS use against source expert use.	Is KBS being used as often as source expert?
		Source Expert	Interface Validity	Validation of KBS interface characteristics with respect to the source expert.	Determine ease-of-use of interface as compared to interfacing source expert
	METHOD	Real System (Source)	Data Collection Veracity	Evaluation of the validity and reliability of test data collected from the source expert.	Identify instrumentation & measurement errors in test data collection.

TABLE 3: IMPLEMENTED KNOWLEDGE-BASED SYSTEM ASSESSMENT

assessments relating to the usability and interface characteristics of the implemented KBS.

Lastly, performing these assessments requires making comparisons between the various aspects of the functional KBS and that of both the source and other nonsource experts. Thus, as was the case with assessment of the elicited KB, the framework in Table 3 includes an additional phase, i.e., assessment of the data collection and experimentation *method* required for making these comparisons. Another consideration here is the use of cross-validation (divergent and convergent evidence). Validation assessments performed utilizing data sets independent of those used for elicitation and system development provide a higher level of assurance than if the same data sets are used for both purposes. Along these same lines, Oren (1985) has suggested that a new type of specialist (a KB auditor), independent from the system developers, is required to audit KBS.

Implemented KBS Verification - The system construction process translates the elicited expertise specification resulting from the KA process into machine executable form. *Verification tests* are therefore necessary to ascertain the correctness of this translation process. These tests can be performed in terms of three criteria: the elicited KB specifications (**implementation verification**), the software and hardware used (**formal checking**), and software engineering requirements (**software quality assurance**). Formal checking techniques include code walkthroughs, syntax checking, rule execution tabulation, and program testing and debugging. Examples of software quality assurance techniques include ensuring modularity in design and requiring proper documentation of the KBS and the tests undertaken.

Assessment of Structure - **Structural assessment** of the KBS utilizes similar techniques and criteria to that of the elicited KB. Comparisons need to be made to both the source expert (**structural validity**) and to other existing nonsource expert sources (**structural comparison**) to determine domain, construct, range, event, primitives, grain-size, abstraction level, and meta-knowledge correspondence. As in the case of the elicited KB, a triangulation of KA methods will be a useful approach for assessing structural validity to the source expert. For example, if the knowledge-structure had initially been elicited utilizing a repertory grid based KA tool such as KSS0 (Gaines, 1987) and ACQUINAS (Boose, 1987), event correspondence can be validated by re-elicitation using a different KA method such as protocol analysis. Similarly, construct correspondence could be validated by using multi-dimensional scaling. Although such triangulation of KA methods is expected to tap into method dependent variances, this does not detract from the fact that by attempting to map a common knowledge source, KA method triangulation is a powerful validation technique. Additionally, triangulation can also be usefully applied to validate KA tools.

Assessment of Behavior - Considering that the same structural assessment techniques can be applied to validating both the elicited and implemented KBs, more comprehensive and stringent assessment at the elicited KB stage can reduce the need for a similar assessment at the implemented stage. This is, however, not the case with behavior assessment. *Behavior assessment* techniques like sensitivity analysis, degenerate tests, extreme-value tests, and model-component tests (Sargent, 1985) take on increased importance here over content-related validity techniques like the logical walkthroughs of the earlier stage. This is because now a functional KBS, or a prototype of it at least, can be used and criterion-related evidence can be collected. The resulting outputs from such tests can be compared to outputs generated by both the source expert (**behavioral validity**) and other nonsource experts (**behavioral comparison**).

Differences in behavior (output) will need to be careful analyzed and traced, as the causes will probably be differences in structure (especially processing logic) or faulty implementation. An example of this are behavioral (output) discrepancies caused by the following gaps (Suwa, Scott and Shortliffe, 1985): conflict (e.g., two rules succeed in the same situation but with conflicting results), redundancy (e.g., two rules succeed in the same situation and have the same results), and subsumption (e.g., two rules have the same results, but one contains additional restrictions on the situations in which it will succeed). Thus, at this implemented KBS stage, not only is behavioral assessment more important, but it can also provide clues as to event correspondence aspects of structural validity as well as defects in implementation (verification). This also highlights the fact that validity assessments should be performed as a co-ordinated and integrated plan of action.

Besides input-output 'black-box' type tests (also likened to the Turing test) that measure correspondence of output behavior under various conditions, it is also important to ensure the *reliability* of these outputs. Thus, goodness-of-fit tests over time will be an appropriate technique.

Assessment of Usability - Two major types of assessment are necessary here. Utilization involves contrasting the frequency of use of the KBS versus the use of the expert (**usage comparison**). **Interface Validity** refers to the ease of use and understandability of the KBS interface. Data for making interface comparisons is obtained by measuring users' perceptions. As reflected in Table 3, both these assessments are made by comparison to the "interaction with a source expert" (i.e., validity).

However, usability assessments can also be made by comparison to some absolute or perceived standards of quality. As these are considered to be part of what is termed in Figure 1 as *functional evaluation*, they are omitted from Table 3 which focuses mostly on validation related assessments. This is also the case with regards to evaluations of the quality and utility of the KBS in relation to its intended application or its domain of applicability (see Definitions section).

CONCLUDING COMMENTS

Validation in a simulation modelling context differs somewhat from validation in KBS development. While simulation models conceal assumptions, structure, and process, KBS attempt to explicitly represent them. As a result, simulation-oriented definitions of validation that are geared towards substantiating a model in terms of performance within its domain of application are inadequate within a KBS context. Similarly, traditional software engineering definitions of validation, as being verification with respect to a series of requirements specifications, are also unsuitable since for KBS such requirements are fuzzy, imprecise and changing (Green and Keyes, 1987). This paper has proposed a representational homomorphism definition of validation that is better suited for validation in the KBS context.

Naylor and Finger (1967) have identified three major methodological (philosophical) positions in the context of validating (verifying) economic and simulation models: rationalism, empiricism, and positive economics. Rationalism holds that any model is simply a system of logical deductions from a series of synthetic premises of unquestionable truth and the validation problem is essentially one of logically assessing these underlying premises. In complete contrast to rationalism is empiricism which regards observation and measurement as the absolute form of validation. Positive economics meanwhile, asserts that predictive behavior assessment is the ultimate form of validation, thus taking a more narrow view of empiricism. Bunge (1983, p. 70) lists

unanimism (intersubjective agreement), pragmatism (success in practice), rationalism, empiricism, critical rationalism (lack of negative evidence) and critical realism as some of the philosophical camps with regard to validation. The validation framework proposed in this paper has accommodated these views, e.g., agreement between experts (unanimism), evaluation as a measure of success (pragmatism), checking logical consistency of rules (rationalism), and comparison of KBS and source experts based on empirical data (empiricism). While the framework brings together these different methodological positions, our view is that *empirical* assessment is central to KA validation. As such the framework identifies the entities to be measured, the type of evidence to be collected, the process which ensures the quality of the data collected, and the type of comparisons to be made to assess validity.

The framework and the methods described in this paper are associated mainly with the validation of the KA process and its component phases. It is however not always possible to disembed validation from the related issues of verification and evaluation. Therefore, the relationships between these concepts are also included in the discussion.

The methods that are discussed provide a portfolio of assessment approaches which are at a higher level of abstraction than those used for a specific KA tool such as KSS0 (Shaw and Woodward, 1988). The development of specific measurement techniques are left to individual investigators who would take into consideration the characteristics of the tools and methods they are dealing with. Nevertheless, the framework proposed could guide developers who wish to validate the KBS they have created and researchers who would like to comparatively assess the quality of various KA tools and methods.

ACKNOWLEDGEMENTS
This work was supported by operating grant OGP2421 from the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Adrion, W.R., Branstad, M.A., and Cherniavsky, J.C. (1982). Validation, Verification, and Testing of Computer Software, ACM Computing Surveys, Vol. 14, No. 2, pp. 159-192.

American Psychological Association (1985). Standards for Educational and Psychological Testing, Washington, DC: American Psychological Association, Inc.

Bachant, J. and McDermott, J. (1984). R1 Revisited: Four Years in the Trenches, AI Magazine, Vol. 5, No. 3, pp. 21-32.

Bobrow, D.G. (1975). Dimensions of Representation, in D.G. Bobrow, and A. Collins (Eds.), Representation and Understanding: Studies in Cognitive Science, New York, NY: AP, 1975, pp. 1-34.

Boose, J.H. and Bradshaw, J.M. (1987). Expertise Transfer and Complex Problems: Using Acquinas as a Knowledge Acquisition Workbench for Expert Systems, International Journal of Man-Machine Studies, Vol. 26, No. 1.

Buchanan, B.G. and Shortliffe, E.H. (1985). Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Reading, MA: Addison-Wesley.

Bunge, M. (1983). Treatise on Basic Philosophy (Volume 6): Epistemology and Methodology II, Dordre Holland: Reidel Publishing.

Chi, M.T.H., Feltovich, P.J., and Glaser, R. (1981). Categorization and Representation of Physics Problems by Experts and Novices, Cognitive Science, Vol. 5, pp. 122-152.

Feigenbaum, E.A. and Feldman, J. (1963). Computers and Thought, New York, NY: McGraw-Hill, pp. 274.

Fellers, J.W.(1987). Key Factors in Knowledge Acquisition, Computer Personnel, Vol. 11, No. 1, pp. 10-24.

Gaines, B.R. (1987). Rapid Prototyping for Expert Systems, in M. Oliff (Ed.), Proceedings of the International Conference on Expert Systems and the Leading Edge in Production Planning and Control, University of Southern California, pp. 213-241.

Gaschnig, J., Klahr, P., Pople, H., Shortliffe, E., and Terry, A. (1983). Evaluation of Expert Systems: Issues and Case Studies, in R. Hayes-Roth, D.A. Waterman, and D.B. Lenat, (Eds.), Building Expert Systems, Reading, MA: Addison-Wesley, pp. 241-282.

Green, C.J.R. and Keyes, M.M.(1987). Verification and Validation of Expert Systems, Proceedings of Western Conference on Expert Systems, Anaheim, California, June 2-4, 1987, pp. 38-43.

Hayes-Roth, R., Waterman, D.A., and Lenat, D.B. (1983). Building Expert Systems, Reading, MA: Addison-Wesley.

Johnson, P.E. (1983). What Kind of Expert Should a System Be?, The Journal of Medicine and Philosophy, Vol. 8, pp. 77-97.

Johnson, P.E., Zualkernan, I., and Garber, S. (1986). Specification of Expertise: Knowledge Acquisition for Expert Systems, Proceedings of 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Oct. 1987, pp. 19.0-19.36.

Naylor, T.H., and Finger, J.M. (1967). Verification of Computer Simulation Models, Management Science, Vol. 14, No. 2, pp. 92-101.

Nunnally, J.C. (1967). Psychometric Theory, New York, NY: McGraw-Hill.

O'Keefe, R.M., Balci, O., and Smith, E.P. (1987). Validating Expert System Performance, IEEE Expert, Vol. 2, No. 4, pp. 81-90.

O'Leary, D.E. (1987). Validation of Expert Systems - with Applications to Auditing and Accounting Expert Systems, Decision Sciences, Vol. 18, No. 3, pp. 468-486.

Oren, T.I. (1985). Artificial Intelligence and Simulation, Proceedings of Artificial Intelligence in Simulation, Ghent, Belgium, pp. 3-8.

Oren, T.I. (1986). Quality Assurance in Cognizant Simulative Design, Proceedings of the 1986 Winter Simulation Conference, Washington, D.C., December 8-10, 1986, pp. 850-852.

Oren, T.I., Elzas, M.S., and Sheng, G. (1985). Model Reliability and Software Quality Assurance in Simulation of Nuclear Fuel Waste Management Systems, Proceedings of the Symposium on Waste Management, Tucson, Arizona, March 24-28, 1985, pp. 381-396.

Quinlan, R. (1987). Addison-Wesley Tapes on Knowledge Acquisition, Addison-Wesley.

Rappaport, A.T. (1987). Cognitive Primitives, Proceedings of 2nd AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Oct. 1987, pp. 15.0-15.13.

Sargent, R.G. (1985). An Expository on Verification and Validation of Simulation Models, Proceedings of the Winter Simulation Conference, pp.15-22.

Sell, P.S. (1985). Expert Systems- A Practical Introduction, New York: McMillan.

Shaw, M.L. and Woodward, J.B. (1988). Validation in a Knowledge Support System: Replication and Consistence with Multiple Experts, to appear in International Journal of Man-Machine Studies: Special Issue on Knowledge Acquisition for Knowledge-Based Systems.

Suwa, M., Scott, A.C., and Shortliffe, E.H. (1985). Completeness and Consistency in a Rule-Based System, in E.H. Shortliffe and B.G. Buchanan, (Eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley.

Todd, P. and Benbasat, I. (1987). Process Tracing Methods in Decision Support System Research: Exploring the Black Box, MIS Quarterly, Vol. 11, No. 4, pp. 493-512.

Vandierendonck, A. (1975). Inferential simulation: Hypothesis-testing by Computer Simulation, Nederlands Tijdschrift voor de Psychologie, Vol. 30, pp. 677-700.

Vedder, R.G. and Mason, R.O. (1987). An Expert System Application for Decision Support in Law Enforcement, Decision Sciences, Vol. 18, No. 3, pp. 400-414.

Yourdon, E. (1986). Managing the Structured Techniques: Strategies for Software Development in the 1990s, New York, NY: Yourdon Press.

Yu, V.L., Fagan, L.M., Wraith, S.M., Clancey, W.J., Scott, A.C., Hanigan, J.F., Blum, R.L., Buchanan, B.G. and Cohen, S.N., (1979). Antimicrobial Selection by Computer: A Blinded Evaluation by Infectious Disease Experts, Journal of American Medical Association, Vol. 242, pp. 1279-1282.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Automated Knowledge Acquisition for a Computer Hardware Synthesis System

William P. Birmingham
University of Michigan
Department of Electrical Engineering and Computer Science
Ann Arbor, Michigan 48109
wpb@caen.engin.umich.edu

Daniel P. Siewiorek
Carnegie Mellon University
Department of Electrical and Computer Engineering
Pittsburgh, Pennsylvania 15213
dps@a.gp.cs.cmu.edu

ABSTRACT

The MICON Synthesizer Version 1 (M1) is a rule-based system which produces a complete small computer design from a set of abstract specifications. M1's design ability depends on the encoding of large amounts of domain knowledge. An automated knowledge acquisition tool, CGEN, works synergistically with M1 by gathering knowledge of how to build and when to use various computer structures. This paper overviews the operation of CGEN by providing an example of the types of knowledge acquired and the mechanisms employed. A novel knowledge-intensive generalization scheme is presented. Generalization is a pragmatic necessity for knowledge acquisition in this domain. A series of experiments testing CGEN's capabilities are presented.

1. Introduction

Knowledge-based systems have gained acceptance in computer-aided design systems for tasks ranging from diagnosis to hardware synthesis. Hardware synthesis, the creation of hardware from a set of high level specifications, is an area of success in the application of knowledge-based system concepts. The success of these systems depends not only on a good formulation of the synthesis problem, but also on the encoding of large amounts of domain knowledge (McDermott, 1981). In some domains, such as computer design, technology evolves so quickly that the knowledge-base of a synthesis tool is never complete. The knowledge-base must be continually enhanced to enable the synthesis system to exploit opportunities afforded by new technology. This phenomenon has shifted the major effort of system development from building problem-solving architectures to developing knowledge-bases. Therefore, automated knowledge acquisition tools are critical to long-term success of knowledge-based synthesis systems.

This work was funded by in part by National Science Foundation grant DMC-8405136 to the Demeter Project, and the Engineering Design Research Center, Carnegie Mellon University, an NSF engineering research center supported by grant CDR-8522616.

The MICON Synthesizer Version 1 (M1) (Birmingham, 1988a) is a rule-based system written in OPS\83 (Production Systems Technology, 1986) which produces a complete small computer design from a set of abstract specifications. During its design process, M1 performs two searches. The first search concerns translating the specifications into physically realizable components. This search process is called the search for function. During the second search, a valid computer structure is formed from components selected in the search for function. The second search is called the search for structure. Both searches occur in large spaces. However, domain knowledge is exploited by M1 to efficiently guide these searches.

An automated knowledge acquisition tool, CGEN (Code Generator) (Birmingham, 1988b), works synergistically with M1 by gathering the knowledge required by M1 to generate a design. CGEN acquires knowledge about how to build and when to use various computer structures[2]. The knowledge is available from hardware designers, domain experts, who are not familiar with the implementation of M1. Since a large number of structures are necessary to build a computer system, CGEN must provide an interface which is convenient to the domain experts. The interface allows designers to express computer structures with schematic drawings and to describe other types of design knowledge (e.g. constraints) without having to write complex programs.

This paper overviews the operation of CGEN by providing an example of the types of knowledge acquired and the mechanisms employed. A novel knowledge-intensive generalization scheme is discussed. Generalization is a pragmatic necessity for knowledge acquisition in this domain. A series of experiments to test CGEN's capabilities are described. However, before the details of CGEN are given, a description of the architecture and knowledge-base of the performance program, M1, is provided. An understanding of M1 is necessary since its behavior is exploited by CGEN to guide the knowledge acquisition process.

2. The Performance Program

The performance program designs small computer systems. These systems are typically composed of the following sub-systems:

- single micro-processor
- static random access memory (SRAM) and read only memory (ROM)
- input/output (IO) devices
- miscellaneous support circuitry (*e.g.* address decoding logic, memory refresh logic)

The complexity of the designs produced by M1 is roughly equivalent to that of an IBM PC-AT.

The input to M1 is a set of high level specifications. The specifications describe global constraints on the design (area, power, and cost) and functional specifications for each sub-system. In Figure 2-1 an example of global constraints and a functional specification for a serial IO (SIO) device are given.

The primitive functions of the M1 architecture consist of search along two axes: a search for function and a search for structure. During the search for function a part implementing the function of a more abstract part must be found. Once the part is found, the search for a computer structure to support it must be performed.

---

[2]Computer structures are legal configurations of hardware, such as the interconnection of a micro-processor to a memory array.

```
Input board  area upper bound:
Input power dissipation upper bound:
Input cost upper bound:

Number of SIO devices [1]:
SIO chip name [?]:
Baud rate for SIO port [300]:
RS232 compatible port [y]:
Port size [25 pin]:
Port type [DCE]:
```

Figure 2-1: Example specifications for global constraints and the SIO sub-system. Default values are given in square brackets.

The search for function occurs along the functional hierarchy. The hierarchy shows how to transform functionally abstract parts into successively more detailed parts until a physically realizable part is found. The functional hierarchy is created by abstracting the function of parts found in the domain. The higher a device is in the hierarchy, the more abstract is its function; similarly, the lower the device is in the hierarchy the better defined is its function. Devices at the lowest point in the hierarchy are referred to as *physical parts*, all others are called *abstract parts*. The use of some form of functional hierarchy is common in design systems which resolve functional specifications. For example, the VEXED system (Mitchell, 1985a) employs a similar tactic for integrated circuit design. A simplified hierarchy is shown in Figure 2-2.

The search for function occurs for a given part, the parent, and the parent's children. Consider, for example, the hierarchy in Figure 2-2 where a search for function might begin with the 68XX_PROC_0. A decision is made between which child, the 6809 or the 6800, to choose. Details of the selection process are described later.

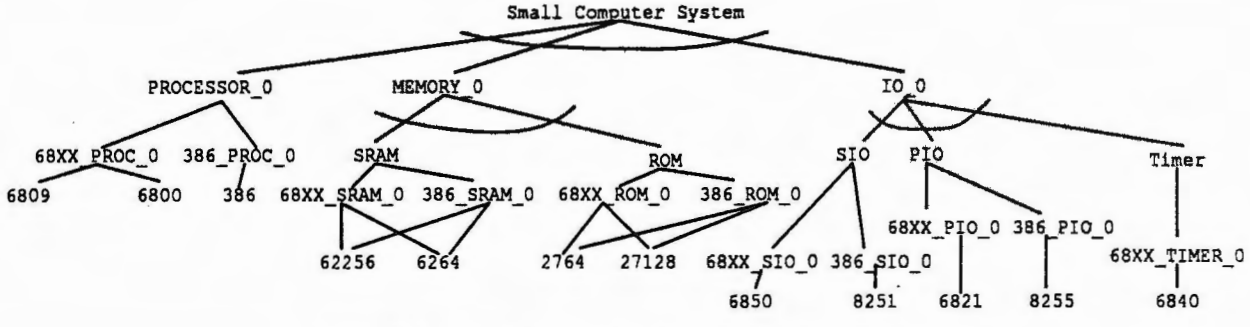


Figure 2-2: Simplified computer functional hierarchy.

Each part in the functional hierarchy has an associated part model. The part model contains, amongst other things, definitions for characteristics and specifications. Characteristics are attribute-value pairs which describe properties of a part. Example classes of properties include: physical, electrical, mechanical, thermal, and physical dimensions. Specifications are also attribute-value pairs which provide

values for properties of less abstract parts into which a given part will be mapped, i.e. values for parts lying below a given part in the functional hierarchy. Specifications, therefore, are excluded from models of physical parts.

Both the functional hierarchy and the part models reside in an external relational database and are fixed before either M1 or CGEN is executed. The database is accessed via a standard query language by M1 and CGEN to retrieve part models.

Once M1 has successfully completed the search for function for a given set of parts, the next phase in the design process is to search for a structure, or mapping, which allows the newly instantiated part to be configured into the evolving design. Templates provide the mapping. An example template is given in Figure 2-3, showing the mapping from 68XX_SIO_0 abstract part into a 6850 physical part. Figure 2-4 shows the *68XX_SIO_0* abstract part -- notice the part has a specific set of signals (represented as pins). These signals, in the context of a full design, are connected to other components. The objective of the mapping is to replace the *68XX_SIO_0* with a *6850* and this associated parts. Figure 2-3 shows how the mapping is accomplished. The box surrounding the *6850* is the *68XX_SIO_0*. The signals connecting to this boundary are more detailed implementations of the 68XX_SIO_0 signals. The other components in the template, the *CLOCK_GENERATOR_0*, *RS232_PORT_0*, and *INTR_BUS_RESOLVER_0* support the operation of the *6850*.

Often, there are multiple techniques for a part to implement its parent's function. Many templates exist in such situations, where each template is associated with an implementation technique. For example, the template in Figure 2-3 maps the 68XX_SIO_0 into the 6850 with an RS-232 external port (as indicated by the part RS-232_PORT_0 in the template). An almost identical template could be used for mapping the 68XX_SIO_0 into the 6850 with an RS-422 external port; the only change necessary is replacing the original RS-232_PORT_0 part with an RS-422_PORT_0 part. Since M1 does not know a template's constituent parts, templates must be marked with pre-conditions to identify when it should be applied. Furthermore, the pre-conditions must be unique across all templates so that no ambiguity exists regarding when a template should be selected. The pre-conditions associated with each template are based upon the *design state*. The design state consists of all reports (a set of variables containing constraint information) and the parts which are involved in the search for function and their corresponding specifications and characteristics. The design state for a parent part P_r and child part P_i is:

$$Design\,State = R_t + (C_r + S_r) + \sum_{i}^{All\,Template\,Parts} (C_i + S_i)$$

Where C_r and S_r are the characteristics and specification of P_r; and, C_i and S_i are the characteristics and specification of P_i. The design state is unique. By associating a template with a design state, unique pre-conditions can be ensured.

The final component of the M1 architecture is the *design cycle*. The design cycle is the technique for using the functional hierarchy and templates to perform design. By iteratively applying the design cycle to all abstract parts, they are refined into physical parts. There are the following five steps (Birmingham, 1988c, Birmingham, 1988a) in the design cycle:

Specification (d_{spec}):
values for the specifications of a part are generated.

Selection (d_{select}): the parts lying below the part to be mapped on the functional hierarchy become

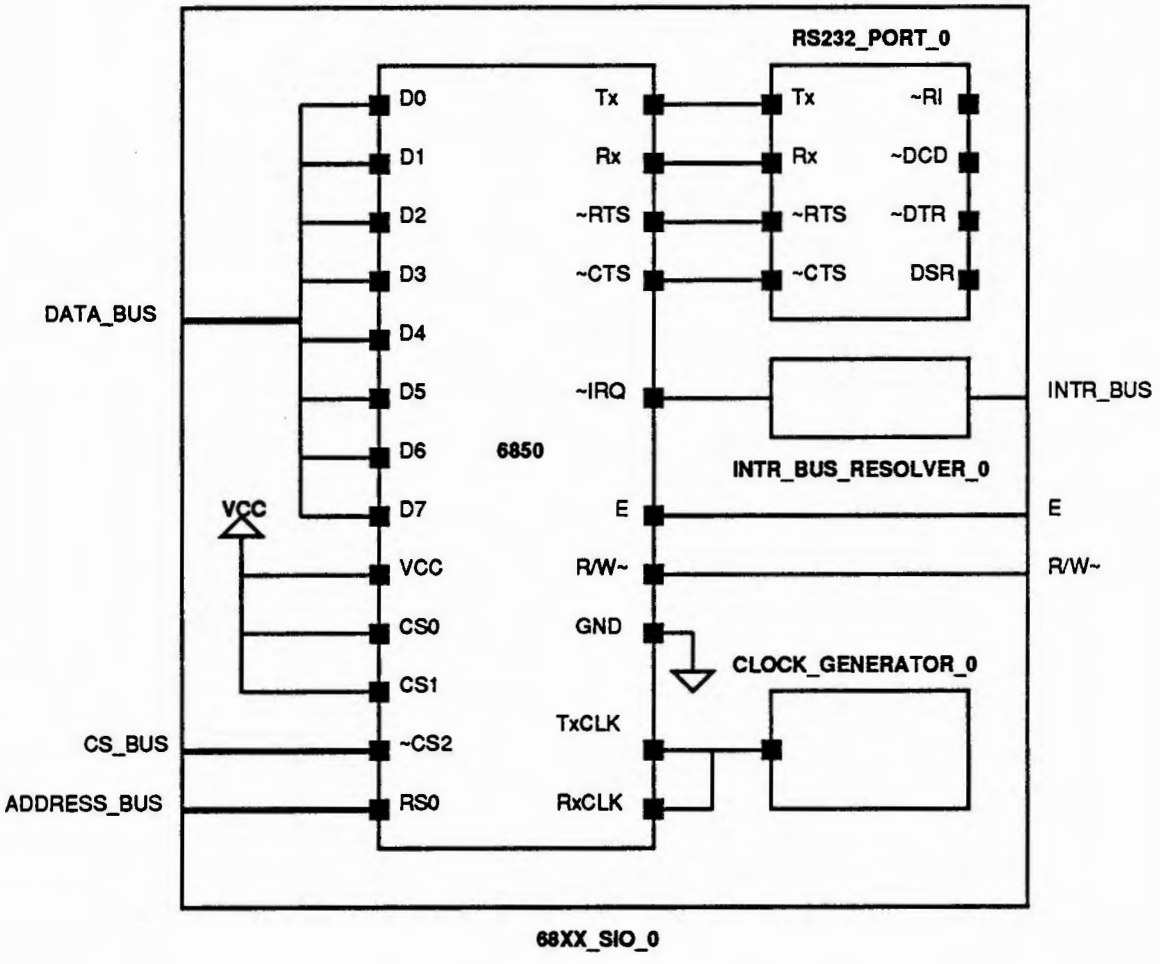


Figure 2-3: Template for mapping a 68XX_SIO_0 into a 6850.

candidates. The characteristic portion of the part model for each candidate is compared to the specifications generated in step d_{spec}. The part which matches most closely is chosen.

Part Expansion (d_{casc}):
cascadable parts, memory for example, are built out to the proper width.

Structure Design ($d_{template}$):
a template is chosen and asserted.

Calculation (d_{calc}): various calculations associated with the template are performed. The calculations consist of updating the values of constraints and generating design information. All the information calculated is stored in a report.

Steps d_{spec} and d_{select} implement the search for function. The search for structure is realized by design cycle steps: d_{casc}, $d_{template}$, and d_{calc}.

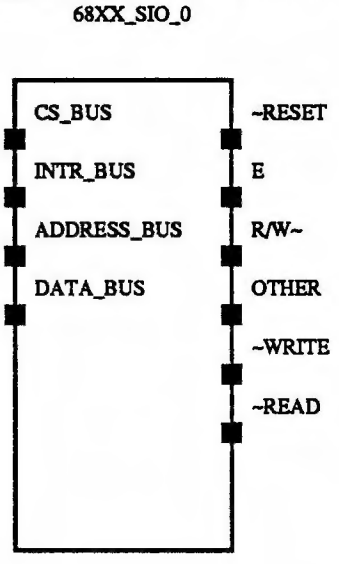


Figure 2-4: The 68XX_SIO_0 abstract part.

A design cycle begins when a part's specifications are filled. Therefore, a design cycle for an abstract part can only begin when its specifications are complete. Spawning of design cycles occurs through a linking of d_{calc} of one design cycle to d_{spec} of other design cycles. During d_{calc} new design information is generated in the form of reports which are then used to complete specifications for some set of abstract parts (which could have been introduced by any template). These parts then begin their own design cycles, causing other parts' specifications to be completed. This action continues until only physical parts exist in the design state.

3. The M1 Knowledge-base

M1's knowledge-base is divided into a set of partitions. Each partition supports either an architectural function or a step in the design cycle. The knowledge partitions are:

Specification Knowledge (k_{spec}):
this partition supports design step d_{spec} by providing values (either by calculation or query) to the specifications of parts in the current design cycle.

Selection Knowledge (k_{select}):
the design step d_{select} is supported by this partition, providing constraints for comparing candidate part characteristics to specifications.

Part Expansion Knowledge (k_{casc}):
this partition supports design step d_{casc}, describing how to build cascadable structures . There are two basic cascadable structures: matrices and trees.

Structural Knowledge ($k_{template}$):
design step $d_{template}$ is realized with this partition through the application of templates.

Calculation Knowledge (k_{calc}):
support for design step d_{calc} is provided by k_{calc}. The means for evaluating design constraints and creating new design information is specified here.

Architecture and Support Knowledge (k_{arch}):
a set of procedures and rules used for implementing the conflict resolution strategy and controlling the execution of design cycles resides in this partition. The remainder of this partition is devoted to a wide variety of functions to support input/output, mathematical functions, etc.

Knowledge contained in partitions k_{select}, k_{spec}, $k_{template}$, and k_{calc} are collectively referred to as *design knowledge.*

A summary of information about the partitions is provided in Table 3-1. Notice that the amount of knowledge in partitions k_{casc} and k_{arch} are fixed over the lifetime of the system. Partition k_{casc} experiences no growth because the structures it describes are generally applicable. M1's problem-solving technique does not change over time, therefore partition k_{arch} will not grow.

Growth Rates and Knowledge Types			
Partition	Procedural	Declarative	Growth Rate
k_{spec}	--	Yes	Rapid
k_{select}	--	Yes	Rapid
k_{casc}	Yes	--	No Growth
$k_{template}$	--	Yes	Rapid
k_{calc}	--	Yes	Rapid
k_{arch}	Yes	Yes	No Growth

Table 3-1: Summary of growth and representation knowledge.

The total knowledge base, K_T, is the summation of the knowledge partitions:

$$K_T = k_{spec} + k_{select} + k_{casc} + k_{template} + k_{calc} + k_{arch} \quad (1)$$

The design cycle imposes dependencies on the partitions. Whenever $k_{template}$ is modified, k_{spec} and k_{calc} must be updated to ensure design cycles are spawned for each part introduced by the template. A *super-partition*, K_T^S, is defined as:

$$K_T^S = k_{spec} + k_{template} + k_{calc} \quad (2)$$

Equation 1 can be re-cast as:

$$K_T = k_{selec} + K_T^S + k_{casc} + k_{arch} \quad (3)$$

CGEN is designed to acquire K_T^S and k_{select}. The remainder of this paper concentrates on the acquisition of K_T^S, with particular attention given to $k_{template}$. Acquisition of k_{select} is a simple extension of the process described for K_T^S.

4. Example of K_T^S Knowledge Acquisition

When M1 can not find a template for mapping a set of parts, it informs the user that a chunk of design knowledge is missing. For example, if M1 requires a template for mapping the 68XX_SIO_0 into the 6850, but one does not exist in its knowledge-base, the following message is generated:

```
Template not found for interfacing the 6850 to 68XX_SIO_0
```

Along with this message, M1 produces a file containing design state information which is used to form a set of pre-conditions for the template to be acquired. An example design state is given in Figure 4-1 corresponding to the template in Figure 2-3. Pre-conditions PC1 through PC7 are specifications for the 68XX_SIO_0, and pre-conditions PC8 through PC12 are global constraints.

As an aside, if the template in Figure 2-3 were modified to create a new template compatible with an RS-422 external interface, pre-condition PC7 in Figure 4-1 would be:

```
(SPECIFICATION DRIVER_TYPE = RS-422);
```

This change would allow M1 to determine precisely when each 68XX_SIO_0 to 6850 template should be used.

```
PC1:  (SPECIFICATION SIO_LINES = 1);
PC2:  (SPECIFICATION PROCESSOR = 6809);
PC3:  (SPECIFICATION TX_BAUD = 1200);
PC4:  (SPECIFICATION RX_BAUD = 1200);
PC5:  (SPECIFICATION TX_TYPE = ASYNC);
PC6:  (SPECIFICATION RX_TYPE = ASYNC);
PC7:  (SPECIFICATION DRIVER_TYPE = RS-232);
PC8:  (REPORT CLOCK_SPEED = 1000000);
PC9:  (REPORT IO_ACCESS_TIME = 500);
PC10:  (REPORT REMAINING_POWER = 10000);
PC11:  (REPORT REMAINING_BOARD_AREA = 500);
PC12:  (REPORT REMAINING_BOARD_COST = 80000);
```

Figure 4-1: Example pre-conditions for template 68XX_SIO_0 to 6850 derived from the design state.

After M1 produces a design state file, CGEN can be invoked once the domain expert has produced a schematic[3], as shown in Figure 2-3, and some other files (described later in this section). This set of knowledge constitutes a training session. The results of the training session are a set of rules, one of which is the $k_{template}$ rule in Figure 4-2. The left hand side (LHS) of the rule provides a generalized version of the pre-conditions from Figure 4-1. The rule's right hand side (RHS) contains a version of the wire list M1 can read.

As specified by K_T^S, values for the specification of all abstract parts introduced by the template must be acquired. In addition, any calculations needed to generate global design information must be culled from the domain expert. To create specifications for a part, the domain expert refers to the part's model and provides a means for generating a value for each specification. Specifications, calculations, and generalization knowledge (described in Section 5) are input via *methods*. An example method for the

[3]CGEN actually uses a wire list generated by the drawing editor from the schematic. A wire list describes how parts are interconnected.

```
RULE instantiate_SIO_template
{
(GOAL name = assert_template)
(SPECIFICATION SIO_LINES = 1);
(SPECIFICATION PROCESSOR =
        (6800 OR 6809));
&A (CHAR MAX_BAUD_RATE);
(SPECIFICATION TX_BAUD < &A);
(SPECIFICATION RX_BAUD < &A);
(SPECIFICATION DRIVER_TYPE = RS-232);
(SPECIFICATION TX_TYPE = ASYNC);
(SPECIFICATION RX_TYPE = ASYNC);
(REPORT IO_ACCESS_TIME > 300);
(REPORT REMAINING_POWER > 200);
(REPORT REMAINING_BOARD_AREA > 500);
(REPORT REMAINING_BOARD_COST > 120);
-->
make (tag name = g8483943)
; The wire list is abbreviated for the sake of conciseness.  Normally
; all parts and connections in the schematic are listed here.
; get_part retrieves a part model from the DB.
get_part(RS232_DRIVER)
get_part(BAUD_RATE_GENERATOR)
...
; connect_net creates a connection between the parts and pins in the
; parameter list.
connect_net(6850, D0, SIO_0, DATA_BUS)
connect_net(6850, TxCLK, BAUD_RATE_GENERATOR, OUT)
...
}
```

Figure 4-2: Final rule for template shown in Figure 2-3.

specification PORT_TYPE for part RS232_PORT_0 is shown in Figure 4-3. A example method for specification CLOCK_FREQUENCY for part CLOCK_GENERATOR_0 is given in Figure 4-4. CGEN ensures that all specifications for all parts have an associated method. If a method for any specification is missing, CGEN requests the domain expert supply it.

```
calculation_type: specification
calculation: PORT_TYPE = MALE;
```

Figure 4-3: Example of domain expert input, a method, for specification PORT_TYPE.

```
calculation_type: specification
calculation: CLOCK_FREQUENCY = TX_BAUD * 16;
```

Figure 4-4: Example of domain input, a method, for specification CLOCK_FREQUENCY. TX_RATE a is specification of the abstract part 68XX_SIO_0.

In addition to methods for specifications, the domain expert also supplies a method for calculating SIO_ADDRESS_SPACE[4]. The method is given in Figure 4-5.

```
calculation_type: calculation
calculation: SIO_ADDRESS_SPACE = 4 ;
```

Figure 4-5: A method for calculating SIO_ADDRESS_SPACE.

5. Generalization

During the creation of a $k_{template}$ rule structural knowledge is joined with a set of pre-conditions. The initial formulation of pre-conditions coincides exactly with the design state, yielding the tightest set of invocation conditions for describing when the new $k_{template}$ knowledge can be applied. This definition is necessary to comply with the M1 architecture's requirement each $k_{template}$ rule have a unique set of pre-conditions.

Often the pre-conditions derived from the design state are overly constraining, preventing the template from being applied in other design states where it is perfectly valid. There are three causes of overly constraining pre-conditions:

Equality: equality is used as the test on a design state variable.

Constants: constants derived from the design state are used for comparison.

Irrelevant pre-conditions:
pre-conditions which are superfluous to the proper definition of the design state are included.

The key to the generalization process is to identify those pre-conditions which exhibit one of these characteristics and implement an appropriate fix. Fixes consist of:

One- or two-sided constraints:
equality test is removed and a single or double-sided interval is used to test a variable.

Variable substitution:
variables are used whenever possible.

Delete pre-condition:
superfluous pre-conditions are removed.

Care must be exercised in the application of fixes to prevent over-generalization, resulting in the intersection of several templates' pre-conditions.

The knowledge needed for generalization, $k_{generalize}$, explains how design variables interact and the proper range of such variables. The $k_{generalize}$ partition is actually composed of two separate bodies of knowledge:

$$k_{generalize} = k_{generalize_rule} + k_{generalize_method} \quad (4)$$

Where each body of knowledge is defined as:

[4]SIO_ADDRESS_SPACE is used by another computer sub-system for building an address decoder.

$k_{generalize_method}$: a description given by the domain expert indicating, for each training case, the variables to relax and their boundary conditions.

$k_{generalize_rule}$: a set of generalization rules, residing in CGEN's knowledge-base, which capture constraint intervals that are applicable to all training cases.

The $k_{generalize}$ knowledge resides with CGEN; it is not part of M1's knowledge-base.

Methods are also used to describe $k_{generalize_method}$ knowledge to CGEN. The methods for relaxing some of the pre-conditions in Figure 4-1 are given in Figure 5-1. The TX_BAUD and RX_BAUD pre-conditions in Figure 5-1 are set less than or equal to the maximum baud rate (MAX_BAUD_RATE)[5] the 6850 can sustain. The IO_ACCESS_TIME is bounded by the access time of the slowest instance of a 6850; a value supplied by the domain expert. Finally, the pre-condition CLOCK_SPEED is removed by the domain expert, because it is subsumed by IO_ACCESS_TIME.

```
method: TX Baud rate max. value
calculation:
 TX_BAUD <= MAX_BAUD_RATE*;

method: RX Baud rate max. value
calculation:
 RX_BAUD <= MAX_BAUD_RATE*;

method: relax IO_ACCESS_TIME
calculation:
  IO_ACCESS_TIME > 300;

method: Delete CLOCK_SPEED
calculation:
  CLOCK_SPEED = DELETE;

*- MAX_BAUD_RATE is a characteristic of the 6850.
```

Figure 5-1: methods for relaxing/deleting pre-conditions.

Figure 5-2 contains an example set of $k_{generalize_rule}$ rules. These rules represent knowledge about general system level constraints. For example, the rule *relax_REMAINING_BOARD_AREA* sums the area consumed by each part asserted by the template (*area_consumed* is a characteristic in every part model). The sum is used as the lower bound for the constraint *REMAINING_BOARD_AREA*.

The $k_{generalize_rule}$ knowledge partition is generated by the *CGEN* knowledge engineer. The $k_{generalize_method}$ knowledge for each training case is examined. Any set of commonly occurring methods are encoded into CGEN rules, thus eliminating the need for these methods to be used by the domain expert in the future. Ideally, after enough training cases are seen, a *complete* set of $k_{generalize_rule}$ rules will exist in CGEN, drastically reducing the number of domain-expert generated methods.

[5]MAX_BAUD_RATE is a characteristic of the 6850.

```
rule 68XX_bus_compatibility
{
...
(SPECIFICATION PROCESSOR = 6809)
-->
(PROCESSOR = 6809 OR 6800;)
}

rule relax_REMAINING_POWER
{
...
(REPORT REMAINING_POWER)
-->
(REMAINING_POWER > total power consumed by template parts;)
}

rule relax_REMAINING_BOARD_COST
{
...
(REPORT REMAINING_BOARD_COST);
-->
(REMAINING_BOARD_COST > total cost consumed by template parts;)
}

{
rule relax_REMAINING_BOARD_AREA
...
(REPORT REMAINING_BOARD_AREA);
-->
(REMAINING_BOARD_AREA > total area consumed by template parts;)
}
```

Figure 5-2: CGEN rules, $k_{generalize_rule}$, for relaxing pre-conditions.

6. The CGEN Architecture

There are two components to CGEN's architecture, the acquisition cycle and rule generation. Two types of acquisition cycles exist, based on the type of knowledge being captured. The cycles are: acquisition of K_T^S and acquisition of k_{select}. The acquisition of k_{select} requires a subset of the functions needed for K_T^S and will not be discussed (for more information see (Birmingham, 1988b)). Rule generation involves the creation of code from CGEN's internal representation.

The function of the acquisition cycle is to guide the overall problem solving activity of CGEN during a training session. The acquisition cycles set up expectations about the types of knowledge being acquired during a training session. For example, when a $k_{template}$ rule is being acquired, CGEN ensures that k_{spec} knowledge is supplied for all parts asserted in the template and that k_{calc} equations are given. The acquisition of these partitions is consistent with the definition of K_T^S.

In conjunction, the acquisition cycle also determines the types of analysis performed on the incoming knowledge. Two types of analysis are performed:

Error/completeness checking:
(applied during both cycles) the correctness and completeness of knowledge acquired during a session is checked. The types of checks used depend on the type of knowledge acquired.

Generalization: (applied during only during K_T^S) as mentioned previously, the pre-conditions associated with a template are generalized.

CGEN, through the acquisition cycle, encodes knowledge of the problem-solving method employed by M1. Implicit in the definition of K_T^S is the relationship of the design cycle (M1's problem-solving paradigm) to the knowledge-base. Since the acquisition cycle is based on K_T^S, it drives CGEN to cull from the domain expert the knowledge necessary to maintain consistency in M1's knowledge-base. This ensures consistent action of the design cycle. Without exploiting knowledge of M1's behavior, CGEN would be incapable of verifying the completeness of incoming knowledge.

The acquisition cycle's definition is also influenced by the nature of the domain knowledge, i.e. the need for generalizing the pre-conditions for $k_{template}$ rules. The behavior of M1 would not be impaired if the generalization process were not applied; however, a burden would be placed on the domain expert to generate a larger number of training cases. For pragmatic considerations, the generalization process is necessary.

During the rule generation process, CGEN's intermediate representation of the acquired knowledge is translated into a set of rules executable by M1. M1 was designed so that all rules in a partition have the same canonical format. The canonical form breaks the LHS and RHS of a rule into two parts: context-specific and instance-specific. All rules in a partition share the same context-specific LHS and RHS. This ensures all rules in a partition are invoked uniformly and have similar actions. The instance-specific portion is generated for each rule based on information gleaned from the training case. For example, for $k_{template}$ rule instances the template pre-conditions form the LHS instance-specific portion of the rule, and the RHS instance-specific portion is created from the wire list. After a training session is complete, the CGEN generated rules are compiled into M1's knowledge-base.

The simplified acquisition cycle for k_P^S is given below:

1. Read methods, wire list, design state, and create intermediate format
2. Identify pre-conditions for $k_{template_rule}$
3. Apply $k_{generalize}$:
   a. Identify overly-constraining pre-conditions
   b. Relax/delete those pre-conditions identified
4. Instantiate $k_{template_rule}$:
   a. Generate LHS from pre-conditions from methods
   b. Generate RHS from wire list from methods
5. Generate k_{calc} rules
6. Generate k_{spec} rules

7. Experimentation

CGEN was subjected to a set of experiments to accomplish the following:

1. test CGEN's ability to capture design knowledge
2. test CGEN's ability to generate working code for M1
3. gain insight into the complexity of the small computer design domain

The experiments were designed to simulate as closely as possible the conditions under which CGEN was intended to perform. A version of M1 with a minimal knowledge-base was used as the performance system. At the beginning of the experimental process, the M1 knowledge-base contained only:

$$K_1 = k_{casc} + k_{arch}$$

Therefore, M1 started with no design knowledge. All subsequent knowledge was acquired through training sessions with CGEN.

M1 was taught to design with the following micro-processor families:

- Motorola 6809
- Motorola 68008
- Motorola 68010
- Intel 80386

A design for each processor was obtained from either published reports or industrial affiliates. All training cases for the physical components were derived from this set of designs. While all the designs were from actual applications, none stretched either the function or performance of any of the processor families.

A group of four designers, *i.e.* domain experts, were used to teach M1. None the domain experts, except one of the authors[6], were familiar with AI programming techniques, the OPS\83 language, or the implementation details of CGEN and M1. The designers received training in M1 and CGEN design philosophy and tool usage roughly equivalent to a two-day course. The domain experts did not write any OPS/83 code or modify the rules created by CGEN[7].

After the designers were familiar with their micro-processor families, the knowledge acquisition process began. The following steps formed the process:

1. part models were developed for all abstract and physical parts
2. data for all part models were entered into the central database
3. templates were drawn for all physical and abstract parts used
4. all methods were prepared
5. knowledge acquisition sessions were conducted using CGEN in conjunction with M1

---

[6]Birmingham entered the 6809 family knowledge.

[7]A call to a routine to generate connections for a certain type of chip was added by the author to one template in each micro-processor family.

7.1. Part Data

The size of the database in terms of part models is given in Table 7-1. Table 7-2 includes the average number of characteristics, specifications (applicable to abstract parts only), and pins per part model.

Part Model Totals	
Abstract Parts	167
Physical Parts	215
Total Parts	382

Table 7-1: Total part models.

Part Model Data		
	Average	Std. Dev.
Specifications/model	4	3
Characteristics/model	4	5
Pins/model	15	18

Table 7-2: Summary of part model data.

7.2. Rule Data

The number of training cases and rules for each design are given in Table 7-3. Table 7-4 shows the breakdown of rules by partition.

When the knowledge acquisition process began, knowledge partitions k_{casc} and k_{arch} were fixed. The k_{casc} partition allowed construction of memory arrays and replication of IO devices; no other structures were necessary. IO replication occurs when the M1 user requests multiple copies of a single function; for example, two SIO devices. In this case, M1 creates as many instances of the function as the user requests. There was no growth in either k_{casc} or k_{arch} during the entire experimentation period, as expected. Table 7-5 shows the average growth rates for all partitions in units of rules per training case.

Training Cases and Rules Per Design		
Design	Cases	Rules
M6809	85	343
M68008	38	173
M68010	19	147
I80386	79	256
TOTAL	221	919

Table 7-3: Number of training cases and rules per design.

The complexity of the LHS of template rules is given in Table 7-6. The first entry in the table is the average size of the design state (or number of template pre-conditions generated by M1 at the beginning of a training session), followed by the average number of pre-conditions which survive to become pre-conditions for a $k_{template}$ rule . The number of pre-conditions removed by the domain expert is given next. This is followed by both the number, and then percentage of pre-conditions generalized of those not

Rules per Partition		
Partition	Rules	% of Total
k_{spec}	552	60
k_{select}	8	.1
$k_{template}$	226	25
k_{calc}	133	15
TOTAL	919	100

Table 7-4: Number of rules per partition.

Average Growth Rate per Partition		
Partition	Growth Rate	Std. Dev.
k_{spec}	2.5	4.8
k_{select}	.04	0.3
k_{casc}	0	0
$k_{template}$	1.0	0.2
k_{calc}	0.6	1.8
k_{arch}	0	0
Number of cases	221	---

Table 7-5: Average growth rate per partition in rules/training case.

deleted. Note the high percentage, 77%, of pre-conditions which are relaxed, verifying that a generalization technique is necessary.

Table 7-7 gives the total number of $k_{generalize_rule}$ rules in CGEN's knowledge-base. The average number of these rules applied during a training session is also given. Finally, the percentage of pre-conditions automatically relaxed and deleted with respect to the data given in Table 7-7 is shown.

The percentage of relaxations and generalizations made by $k_{generalize_rule}$ indicates the coverage of these rules with respect to all the generalization knowledge required for a training session. Note that the number of training cases utilizing the generalization rules (given by the number of cases entry in Table 7-7) is less than the total number of training cases in Table 7-3. The generalization rules were added to CGEN's knowledge base late in the experimental process.

The RHS data for $k_{template}$ rules is summarized in Table 7-8. The table shows the number of wires and parts asserted per rule. The large number of wires relative to number of parts is explained by three factors. First, many templates are comprised of wires only, implementing a *switchbox* function. Second, many templates (especially for the 80386) require a large number of wires to connect the parts in the template. Consider showing the connections for an 80386 which has 32 address and 32 data lines. Finally, the templates tend to have local information which does not require a large number of support parts. Locality of knowledge indicates the incremental nature of template knowledge. Templates divide the structural design problem into relatively small chunks.

(a) INITIALLY, SELECT THE MODULE "SYSTEM" AS FOCUS MODULE.

(b) WHEN THE FOCUS MODULE IS BELIEVED TO BE BROKEN,
SELECT A NEW FOCUS MODULE FROM THOSE WHICH ARE SUB-MODULES.

(c) WHEN THE BELIEF STATE CHANGES,
SELECT A FOCUS MODULE FROM THE MODULES THAT ARE MOST DEEP
FROM THE MODULES THAT ARE MOST SUSPECT.

(d) THE FOCUS MODULE IS SELECTED WITH REPLACEMENT.

Figure 13 — Strategy Articulation for Opportunistic Module Selection

(4) Any time sufficient elements are arrayed on the table, the work can begin by processing or combining the elements to produce new elements. The result of ANY processing operation can require selection of a new element from any bucket.

(5) When a new element is selected from the bucket, the process has implicitly backed up, and several other elements may need to be selected.

The selection and backup process, then, is centered around conceptual objects that are needed in constructing a solution, not around the individual clauses of the procedure, as in Prolog. Backtracking, then, is not the step-by-step chronological process through each atomic operation, but a rejection of one of the selections. While not strictly non-monotonic in nature, this model does share some of the characteristics of dependency-directed backtracking.

5. SCOPE OF THE METHOD

We do not yet have an accurate assessment of the scope of this approach. We feel roughly that inference strategies can be divided into four types.

Our rough characterization of inference strategies is that they fall along two dimensions:

(a) Is the strategy one of selection or construction?
(b) Is the strategy monotonic or non-monotonic?

A strategy is monotonic if elements from the solution space are considered in such a way that an element is either accepted or rejected before the next solution space element is considered. In other words, each proposed element is considered on its own merits, without comparison to the merits of alternative members of the solution space.

A strategy is non-monotonic if more than one element of the solution space is selected or constructed for consideration at a time.

In a selection strategy, the solution set consists of a pre-enumerated set of answers. The solution set may, of course, be organized along one or more hierarchical classifications. For example, in a typical troubleshooting problem, the solution set is the set of replaceable units, which are typically organized hierarchically into modules and sub-systems (according to function) and also into boards and cages (according to physical arrangement).

In a construction strategy, members of the solution set are not specified explicitly but are built as needed from a collection of components representing objects and relationships. Plans, engineered artifacts and even detailed temporal explanations are examples of such constructions. In general, the components can be used over and over again, so that the solution set is potentially countably infinite.

This leaves us with four basic classes of inference strategies:

1. MONOTONIC SELECTION STRATEGIES - solution set members and/or their organizing classes are selected for consideration in some appropriate order. Once a member or class is selected, an immediate determination of its validity is made. If an element is rejected as invalid, it is not reconsidered.
2. MONOTONIC CONSTRUCTION STRATEGIES - the solution is constructed piecemeal by selecting components and deciding whether and how to include them in the current partial solution. This decision is made without reference to other partial solutions. Each component is selected by a monotonic selection strategy. Once a component is rejected for inclusion in a partial solution it is not later reconsidered for inclusion in the same partial solution.
3. NON-MONOTONIC SELECTION STRATEGIES - multiple selection paths through the various organizing hierarchies are maintained at once. At any point in time, the choice can be made to (a) extend an existing selection path, (b) start a new selection path along some dimension, (c) reject a path, or (d) merge two paths via an appropriate union or intersection operation.
4. NON-MONOTONIC CONSTRUCTION STRATEGIES - partial solutions are maintained. Components may be selected and combined to produce new partial solutions. At any point in time the choice can be made to (a) select a component and partial solution for possible combination, (b) start a new partial solution, (c) reject a partial solution, or (d) combine two existing partial solutions to form a new one.

Our best guess is that our method of knowledge-mediated control works quite well for monotonic selection strategies and even most monotonic construction strategies since many construction strategies are organized around a search for components to add to the construction. We do not as yet have a feel for whether non-monotonic strategies can be adequately captured by this approach. Non-monotonic strategies seem to require massive global mechanisms, such as blackboards, truth-maintenance, and constraint propagation networks. We intend to investigate the connection more thoroughly in the future.

6. CONCLUSIONS

In this paper we have demonstrated a method for designing inference engines. The entire analysis and design process is governed by two major principles. The principle of ontological analysis dictates that all knowledge structures and operations used in the inference process be based on articulations of the task experts. The principle of knowledge-mediated control dictates that any deviation from random selection in the inference process be organized by a knowledge structure or local operation.

If these principles are adhered to, the knowledge engineer can be reasonably confident of a system where the knowledge acquisition and maintenance costs are not excessive. HIPE, an inference engine designed according to these principles (Freiling et al., 1986, Rehfuss et al., 1985). has been successfully fielded in several applications for telecommunications system troubleshooting where technicians have built the knowledge bases without intervention by trained AI specialists. A knowledge acquisition tool, ASTEK (Jacobson and Freiling, 1987). has been developed and is now in evaluation for fielded use.

Our current effort is aimed at refining the inference model presented in section 4, and building a tool that allows practicing knowledge engineers to develop and test inference strategies without having to program the strategies.

The methodology we have illustrated for troubleshooting is applicable to other problems as well. Although it is still informal, we believe it provides a powerful tool for beginning knowledge engineers to proceed step-by-step from an analysis of expert language to prototype inference engine designs.

APPENDIX — THE ONTOLOGY OF ELECTRONIC INSTRUMENT TROUBLESHOOTING

This appendix shows the ontology from which our inference engine design proceeds. Quite briefly, domains are constructed from other domains by a small set of construction functions:

{a, b, c}	the domain consisting of atomic elements a, b, and c.
D + E	the domain consisting of each member of D and each member of E.
D × E	the domain consisting of all ordered pairs whose first element is a member of domain D and whose second element is a member of domain E.
D → E	the domain consisting of all total functions that map members of D into members of E.
2 ** D	the domain consisting of all subsets of D.
D *	the domain consisting of all ordered sequences of elements of D.

Static Ontology

NODE = { node1, node2, ... }
KNOB = { power button, volts_per_div_knob, ... }
SETTING = { on, off, ... } + INTEGER
COMPONENT = RESISTOR + TRANSISTOR + ...
RESISTOR = { resistor1, resistor2, ... }
TRANSISTOR = { transistor1, transistor2, ... }
TERMINAL = { base, collector, emitter, left_end, right_end, ... }
BLOCK = { power_supply, sweep_circuit, trigger_circuit, ... }
MODULE = BLOCK + COMPONENT + SETTING
STIMULUS = VOLTAGE + WAVEFORM
WAVEFORM = WAVE_TYPE × FREQUENCY × AMPLITUDE
WAVE_TYPE = { sine_wave, square_wave, triangle_wave, ... }
FREQUENCY = INTEGER
AMPLITUDE = VOLTAGE
VOLTAGE = INTEGER

SETTINGS = KNOB → SETTING
CONNECTIONS = (KNOB × TERMINAL) → NODE + { unassigned }
STIMULI = NODE → STIMULUS + { left_alone }
CONTENTS = MODULE → 2 ** MODULE

replaceable: MODULE → { replaceable, composite }

Dynamic Ontology

INSTRUMENT_STATE = CONNECTIONS × SETTINGS × STIMULI
OBSERVATION = VOLTAGE + WAVEFORM

ADJUSTMENT = REMOVE + SET
REMOVE = { remove } × COMPONENT
SET = { set } × KNOB × SETTING
MEASUREMENT = { measure } × NODE

perform_adjustment:
    (ADJUSTMENT × INSTRUMENT_STATE)
        → INSTRUMENT_STATE
perform_measurement:
    (MEASUREMENT × INSTRUMENT_STATE)
        → OBSERVATION

Epistemic Ontology

BELIEF_STATE = MODULE → BELIEF
BELIEF = { ok, broken, suspect , unknown }
TROUBLESHOOTING_RULE =
    ADJUSTMENT × MEASUREMENT ×
    EXPECTATION × CONCLUSION
EXPECTATION = OBSERVATION
CONCLUSION = MODULE × BELIEF

expectations_fulfilled:
    (EXPECTATION × OBSERVATION)
        → { fulfilled, not_fulfilled }
assimilate_conclusion:
    (CONCLUSION × BELIEF_STATE) → BELIEF_STATE
apply_rule:
    (TROUBLESHOOTING_RULE
        × INSTRUMENT_STATE × BELIEF_STATE)
            → (INSTRUMENT_STATE × BELIEF_STATE)
diagnosis_finished:
    BELIEF_STATE → { done, not_yet_done }

REFERENCES

Alexander, J.H., Freiling, M.J., Shulman, S., and Staley, J. (1986). Knowledge Level Engineering: Ontological Analysis, *1986 National Conference on Artificial Intelligence*, AAAI, Philadelphia, PA, August 11-14, 1986.

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*, Vol. 26, No. 4.

Cantone, R.R., Pipitone, F.J., Lander, W.B., and Marrone, M.P. (1983). Model-Based Probabilistic Reasoning for Electronics Troubleshooting *Eighth International Joint Conference on Artificial Intelligence*, Karlshruhe, W. Germany, August 8-12, 1983.

Clancey, W.J., (1986). Know-how vs. Knowledge Representation, *1986 Workshop on Knowledge Compilation*, Oregon State University, The Inn at Otter Crest, OR, September 24-26, 1986.

Cohen, P.R., (1985). Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach, Pitman Publishing, Inc., Marshfield, MA, 1985.

Freiling, M.J., Alexander, J.H., Feucht, D., and Stubbs, D., (1984). GLIB -- A Language for Describing the Behavior of Electronic Devices, *Applied Research Technical Report CR-84-12*, April 6, 1984, Tektronix, Inc., Beaverton, OR, April 6, 1984.

Freiling, M.J., Alexander, J.H., Messick, S.L., Rehfuss, S., and Shulman, S., (1985). Starting a Knowledge Engineering Project -- A Step-by-Step Approach, *A.I. Magazine*, Vol. 6, No. 3.

Freiling, M.J., Alexander, J.H., Messick, S.L., Rehfuss, S., and Shulman, S., (1986). The Ontological Structure of a Troubleshooting System for Electronic Instruments, *Applications of*

Artificial Intelligence in Engineering Problems, D. Sriram and R. Adey (Eds.), Springer-Verlag, New York, NY, 1986.

Freiling, M.J., (1988). Designing an Inference Engine: From Ontology to Control,*1988 International Workshop on Artificial Intelligence for Industrial Applications*, IEEE, Hitachi City, Japan, May 25-27, 1988.

Gruber, T. and Cohen, P., (1986). Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition, *1986 Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Banff, Canada, November 3-7, 1986.

Jacobson, C. and Freiling, M.J., (1987). ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge, *1987 Workshop on Knowledge Acquisition for Knowledge-Based Systems*, AAAI, Banff, Canada, October 18-23, 1987.

Phillips, B., Messick, S.L., Freiling, M.J., and Alexander, J.H., (1985). INKA: The INGLISH Knowledge Acquisition Interface for Electronic, Instrument Troubleshooting Systems, *Applied Research Technical Report CR-85-04*, Tektronix, Inc., Beaverton, OR, January 7, 1985.

Rehfuss, S., Alexander, J.H., Freiling, M.J., Messick, S.L., and Shulman, S.J. (1985). A Troubleshooting Assistant for the Tektronix 2236 Oscilloscope *Applied Research Technical Report CR-85-34*, Tektronix, Inc., Beaverton, OR, September 25, 1985.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

KNOWLEDGE ACQUISITION AND TECHNOLOGY

Brian R Gaines
Knowledge Science Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4
gaines@calgary.cdn

ABSTRACT

A model of knowledge-acquisition for knowledge-based systems is developed which presents the acquisition activity as playing an essential and continuous role in skilled performance, rather than as a separate and separable activity. The practical implications of this model for systems design are developed, and recommendations made targeted on monitoring the quality of advice from expert systems and achieving closer integration between the application of these systems and the formation of expertise. The model is developed in depth to generate taxonomies of human knowledge processes and use these to analyze the roles of a wide variety of computer-based systems in supporting these processes. The model is used to highlight strengths and weaknesses in the current state of the art in knowledge representation. This paper provides an overall framework for the variety of knowledge acquisition problems, techniques and technologies discussed in the literature.

INTRODUCTION

One of the most interesting aspects of research on knowledge acquisition is the sheer diversity of topics and approaches that are reported: from cognitive models of human behavior to computer tools for knowledge elicitation; from interviewing techniques to text analysis; from practical experience of expert system development to laboratory studies of the consistency of expertise; from narrow domains of particular systems to broad requirements for the integration of numeric computing, simulation, databases, communications and hypermedia. However, this diversity is also confusing. What is the underlying cohesive theme of 'knowledge acquisition.' Does the diversity reflect the sheer anarchy of human knowledge processes as Feyerabend (1975) suggests, or is there some reasonable infrastructure in common to the different aspects of knowledge acquisition? Is not the whole of computing in the support of knowledge and, if so, what is special about knowledge-based systems and knowledge acquisition?

At the first AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems in Banff, November 1986, I proposed that the cognitive entity to be considered in studying knowledge acquisition is human society rather than individuals (Gaines 1987a). Society acts as a distributed anticipatory system modeling and controlling the world to enhance its probability of continued existence. Knowledge transfer processes within society are manifestations of coordinating communications between the components of the distributed system. A taxonomy of such processes was developed and used to classify various approaches to the computer-based support of knowledge acquisition.

At the first European Workshop on Knowledge Acquisition for Knowledge Based Systems in Reading, September 1987, I proposed that expertise formation is a manifestation of social processes encouraging functional differentiation in the, otherwise fairly uniform, distributed anticipatory system (Gaines 1987b). Our culture provides a wide variety of positive feedback mechanisms, corresponding to the taxonomy of knowledge transfer processes, that: support individuals in the development of excellence; discourage uniformity in specialization; and encourage the sharing of specialist knowledge.

At the second AAAI Workshop in Banff, October 1987, I analyzed the techniques and technologies being developed for knowledge-based systems and knowledge acquisition (Gaines 1987c). An overall taxonomy of integration issues in knowledge-based systems was developed encompassing person-computer interaction, the roles of experts, clients and knowledge engineers, related computing technologies, and the shell, support environment and knowledge acquisition tools. This was used to generate complementary design requirements at each branch in the taxonomy, one relating to integration issues and the other to separation issues.

At the second European Workshop in Bonn, June 1988, I described the developmental trends in knowledge acquisition systems: from manual techniques in the 1970s; through a first generation of computer-based tools in the early 1980s; to a second generation of integrated knowledge support systems in the late 1980s (Gaines 1988a). These trends were analyzed in terms of the socio-cultural model described above, and in terms of their technical foundations through a model of the infrastructure of generations of information technology. A basic taxonomy was developed of the variety of human knowledge processes, the underlying techniques involved, and pre- and post-computer supporting technologies.

This paper attempts to provide an integrative framework for knowledge acquisition for knowledge-based systems that has deep foundations but is also clearly applicable to the technology and systems development.

THE EMBEDDING OF KNOWLEDGE ACQUISITION IN EXPERT ACTIVITY

The distinction between knowledge acquisition and performance systems is an invidious one. It can be taken to imply that expertise is a static collection of skills and knowledge, capable of being elicited and transferred to an expert system that will then be able to reproduce the expertise. This static model is an extremely dubious one in systems design, leading to systems that date rapidly, interfere with the growth of expertise, have poor reliability and high maintenance costs. The following model explains how these conditions arise, and leads to approaches for avoiding them.

Hawkins (1983) has abstracted from industrial experience in developing mineral exploration expert systems and proposed a model of human expertise relevant to expert systems. Figure 1 presents a summary of the essential features of this model:

- The expert first elicits data about the problem from the client;
- He or she develops a minimal model that accounts for the data provided;
- He or she generates advice based on the model and feeds this back to the client;
- The client may accept the advice, or query it and, possibly, the model;
- The queries lead to further data elicitation, and repeat of the elicitation/modeling/advice/query cycle.

Thus, in Hawkins' model, the client plays an active role in further developing the model by providing more data until he or she is satisfied with the model and consequent advice. Expert advice giving and taking is part of a cycle of negotiation around a process of model formation.

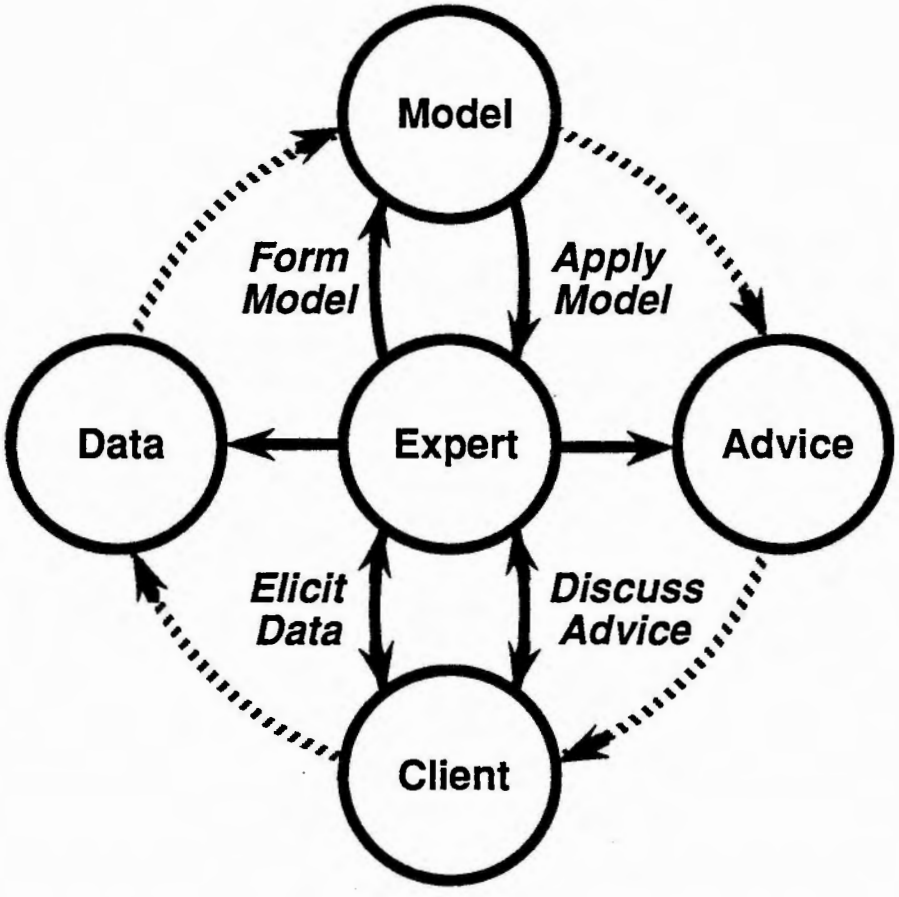


Fig.1 The negotiation cycle in expert-client interaction

The particular importance of this model of expert performance is the emphasis it places on the embedded knowledge acquisition process. The expert is managing a data collection, modeling and decision cycle within a consultation. As Figure 1 shows, this implies that the expert is able to enhance his or her collection of case histories, models and advisory strategies through the consultation. Skilled performance is not the static utilization of expert knowledge and skills in Hawkins' model—it is instead a microcosm, and an essential part of, the wider processes of knowledge acquisition which are intrinsic to the role of an expert.

EXTENDING ACQUISITION THROUGH THE PROFESSIONAL COMMUNITY

Access to the learning experiences of attempting to solve the problems of a client community is one major source of knowledge in the development of the expertise of an individual expert. The client community provides access to practical experience, a variety of problems, particularly novel ones that go beyond existing expertise, and it also manages the growth of expertise through systems of reward, criticism and access to resources. However, learning from experience is a slow and error-prone process, and socially significant areas of expertise become associated with professional communities that attempt to expedite learning and reduce errors through the sharing of experience. Figure 2 extends the model of Figure 1 to include these other processes, such as apprenticeship, instruction, training, education, workshops, conferences, journals and books. Professional communities usually also play a major role in refining and directing the client community's reward, criticism and resource allocation systems.

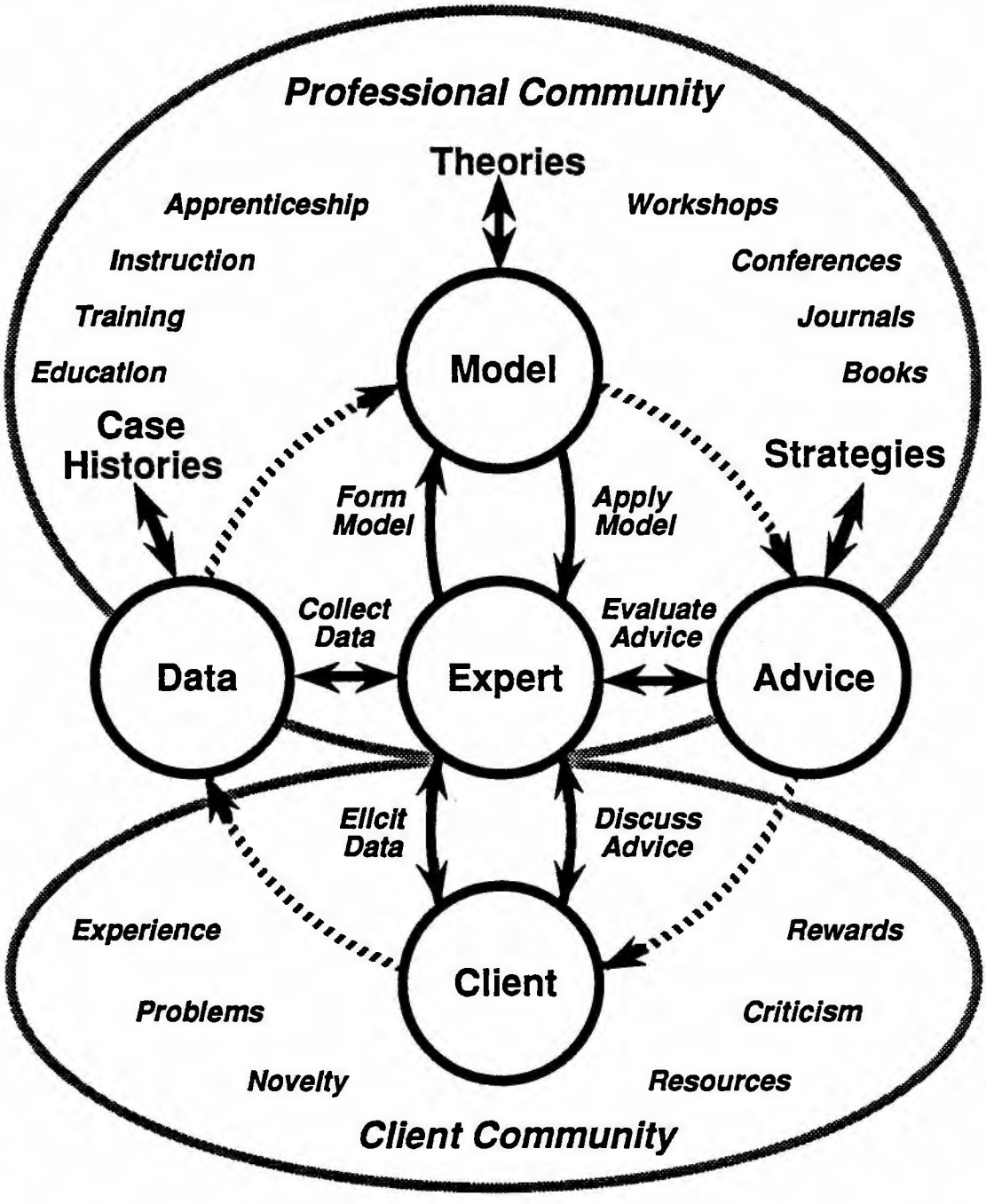


Fig.2 Extended knowledge acquisition process in expert-community interaction

The role of the professional community can be seen as an indirect enhancement of the basic processes of knowledge acquisition in the negotiation cycle of expert performance. The individual can acquire the data of others through access to reported case histories. He or she can enhance his or her repertoire of models through the published theories developed by others to account for their data and the data of many others. He or she can enhance his or her repertoire of advice through the published advisory strategies developed by others and evaluated by them, or yet others, against past case histories, theoretical foundations, and experience in use.

This model of individual expertise as intrinsically a knowledge acquisition process at multiple levels intrinsically embedded in, part of, and supported by essential social processes, is consistent with Stich and Nesbitt's (1984) view of the social role of experts as managers of the *reflective equilibrium* of the inductive process of knowledge generation in society. It is also consistent with the emphasis in professional development books concerned with good practice on the importance of balancing experts' authoritarian roles as knowledge repositories and appliers with their entrepreneurial roles as knowledge acquirers (Schön 1983).

Sociologists have noted the very strong positive feedback processes in the dynamics of knowledge acquisition in the scientific community (Hagstrom 1965). Merton (1973) coined the term the "Matthew effect" for those features of the reward system in research that were biased towards allocating greater credit for the same discovery to those with an already established reputation. The qualitative effect of such positive feedback processes is to amplify random differences within a population, creating strong distinctions which have no other basis than the feedback process itself. That is, even if all scientists were created equal in their capabilities, some would become experts relative to others because random differences in early performance generated credit that affected resource allocation and gave them greater opportunities to acquire knowledge. This strength of this effect can be demonstrated by studies of the behavior of 'experts' simulated by learning automata in a competitive learning environment with access to resources related to demonstrated 'expertise' (Gaines 1988b).

In many areas of expertise where expert systems are expected to play a major role these positive feedback processes are exceptionally strong. In medicine, for example, the key learning resource is access to medical problems, and the 'owner' of such a problem has a strong personal interest in only allowing someone of very good reputation to handle it. The system, including considerations of legal liability, strongly funnels problems to those who are regarded as experts. It is, however, precisely these problems from which new knowledge is generated. Hence direct experience of case history data is made available primarily to those with already established reputations for having capabilities to deal with it. Similar considerations apply to the award of scholarships, invitations to scientific congresses, and so on (Blume 1974). They also apply not only to individuals but also to social units, such as a company subject to government procurement procedures that are heavily biassed to contractors with prior experience and with whom the government agency has prior experience.

IMPLICATIONS OF SOCIAL ACQUISITION MODELS FOR SYSTEM DESIGN

The model of the expert's relation to those seeking advice as one of trading it for access to problem-solving, and hence learning, opportunities is also consistent with the actual behavior patterns of experts in relation to expert systems. Divesting the unprofitable trading to focus on the profitable is a sensible strategy, that is, passing to an expert system those standard problems where little learning opportunity is expected. However, the uncertainties about the classification of any problem situation suggest there may be dangers in this approach which could be avoided by adequate design and application of knowledge-based systems. The advice giving system should contain rules which indicate problem-solving situations that are beyond the boundaries of its capabilities and refer these to a human expert (in the same way that human experts refer difficult problems to others). It should log actual case histories of advice giving for later review by human experts so that these rules can be validated. It should make provision for feedback as to the results of advice so that learning opportunities are not lost.

The continuity between the modeling role adopted by the expert in Hawkin's model of expert-client negotiation, and the expert's wider modeling role in managing the inductive inference process, suggests that the performance and knowledge acquisition phases of expert systems should not be separated as they are currently. It would be better to think in terms of *knowledge support systems*

that integrate knowledge acquisition and advisory roles. In particular, the model of system-user relations as mutual negotiation rather than authoritarian advice giving is a good model for knowledge-based system.

The emphasis on the natural unit representing expertise being a social one rather than an individual emphasizes the need for knowledge-based system development to be based on groups of experts rather than individuals. Figure 2 illustrates the diversity of sources, models and strategies likely in such a group and emphasizes the unlikelihood of there being a complete *consensus* in the knowledge used by the experts. The ongoing acquisition process makes consensus not only unlikely but also undesirable since competing, possibly mutually contradictory, interpretations, models and strategies are an essential feature of active knowledge development. It is important to develop knowledge-based systems that can encompass multiple sources of expertise that may be mutually conflicting, and make effective use of the conflict, for example, to generate 'dissenting' opinions that indicate the boundaries of the expertise (Boose & Bradshaw 1987). Knowledge acquisition systems must be able to elicit, make explicit, and make effective use of different conceptual systems and terminologies (Shaw & Gaines 1988).

All these considerations are also relevant to questions about the accountability of expert, expert system, and client for the advice. However, that is a side-effect of the social pressures on the inductive process—accountability, and the resultant rewards and punishments, are cultural mechanisms to encourage learning and the appropriate use of expertise. Well designed expert systems will make provision for the clear presentation of the quality of advice, reference to human experts, audit trails, logging, and case history review, because these are necessary to an effective technology—the possible legal consequences of not doing this are just an indication of social expectations for an effective technology in the domain of knowledge acquisition and transfer.

There are implications for the detailed technical design of knowledge-based systems. As an example consider the form of rule often induced from case data which states that if the value of an attribute is unknown it is reasonable to fill in it with a default value based on the value of other attributes; for example, if the induction algorithm relates the value of attribute C to that of attribute A:

If (A Equal B) And (C Equal Unknown) Then (Set C Equal D)

This rule uses known data and induced relations to make suppositions. It is also reasonable, but unusual, to add a meta-rule based on the same relation:

If (A Equal B) And (C Not-Equal D) Then (Set Anomaly Equal True)

This meta-rule uses known data and induced relations to highlight anomalies (unexpected data). The new attribute 'Anomaly' indicates that the value of C is inconsistent with previous data used in inductive inference. It can be used to trigger query or alarm reports, and the normal explanation mechanism based on back-tracking will trace the anomaly to its source (Rappaport & Gaines 1988). It can be used to indicate that the problem under consideration is unusual and should be referred to a human expert or, at least, be recorded and flagged of interest.

In this example, the conflict between the rule and the data is exactly the reflective equilibrium managed by experts noted above. The meta-attributes generated through anomaly-detection rules may be used as a basis for machine learning, as Lenat (Davis & Lenat 1982) has done in AM (terming anomalies 'interesting'). However, they are also a basis for effective integration between expert and expert system, with the human expert continuing to have the role of managing the inductive process.

SOCIO-COGNITIVE FOUNDATIONS OF KNOWLEDGE PROCESSES

The extended knowledge acquisition process model of Figure 2 is attractive in capturing much of what we know about expertise and its role and formation in society and the individual. It suggests a number of significant design considerations in the development of knowledge-based systems as discussed in the last Section. However, it is not detailed enough to act as a design framework in its own right. This Section develops the model in greater deal and then uses it to further discuss the role and development of knowledge-based systems.

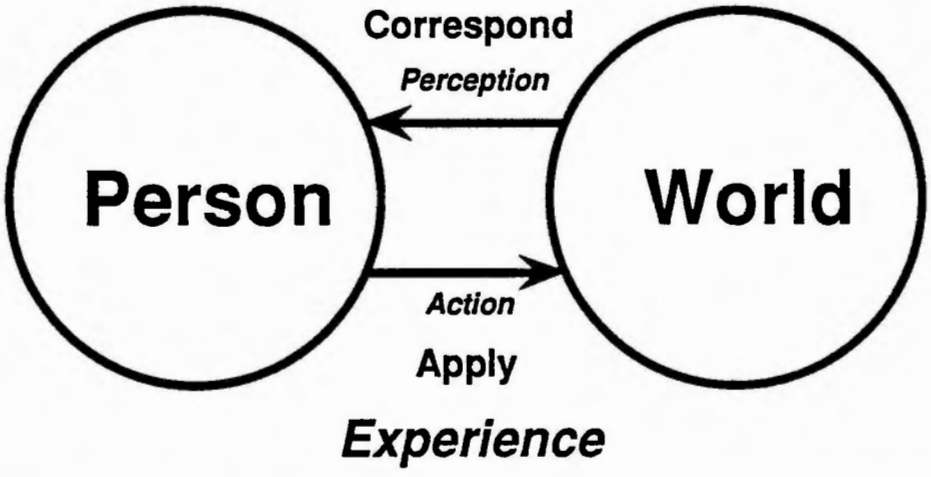


Fig.3 Person as an anticipatory system interacting with the world

Figure 3 shows the basic cognitive unit normally considered—the individual as an anticipatory system, experiencing the world, perceiving it, acting upon it, modeling it to predict and explain it, interacting with it to experiment with and control it. The cognitive structures developed are valid in so far as they correspond to, or apply to, the world. I have previously used a hierarchical structure based on Klir's (1985) modeling theory to analyze the activities of the individual in coming to terms with the world (Gaines 1987a). Figure 4 shows a variant of this hierarchy emphasizing the cognitive processes involved in modeling the world and their definitions in abstract, systemic terms:

- To *recognize* at the lowest level is the capability to notice recurrence of the 'same' events in the world when they recur. This is already a significant cognitive act because 'same' is subject to personal definition, and the concept that events recur is a strong presupposition. Recognition is fundamental to any modeling system but is in itself a weak operation since it is dependent on the recurrence to make use of the data.
- To *recall* at the next level is the capability to regenerate the distinction used in recognition internally so that it is itself an 'event' that may be processed. This facility to recreate events in the 'imagination' is fundamental to the existence of the cognitive process, detaching human knowledge processing from the immediacy of experience.
- To *model* at the next level is the capability to derive the distinction used in recognition and recall from other distinctions that may themselves not relate directly to experience. This facility to 'explain' events in terms of 'deeper' distinctions that only indirectly relate to experience is again fundamental to the efficiency of the cognitive process, allowing novel distinctions to be developed that efficiently encode wide ranges of otherwise unrelated experience.
- To *reconstrue* at the next level is the capability to derive one distinction from multiple models. This facility to move between modeling systems is fundamental to the adaptability of the cognitive process, and the human species, allowing a wide repertoire of anticipatory sub-systems to be developed to cope with the variety of the world.

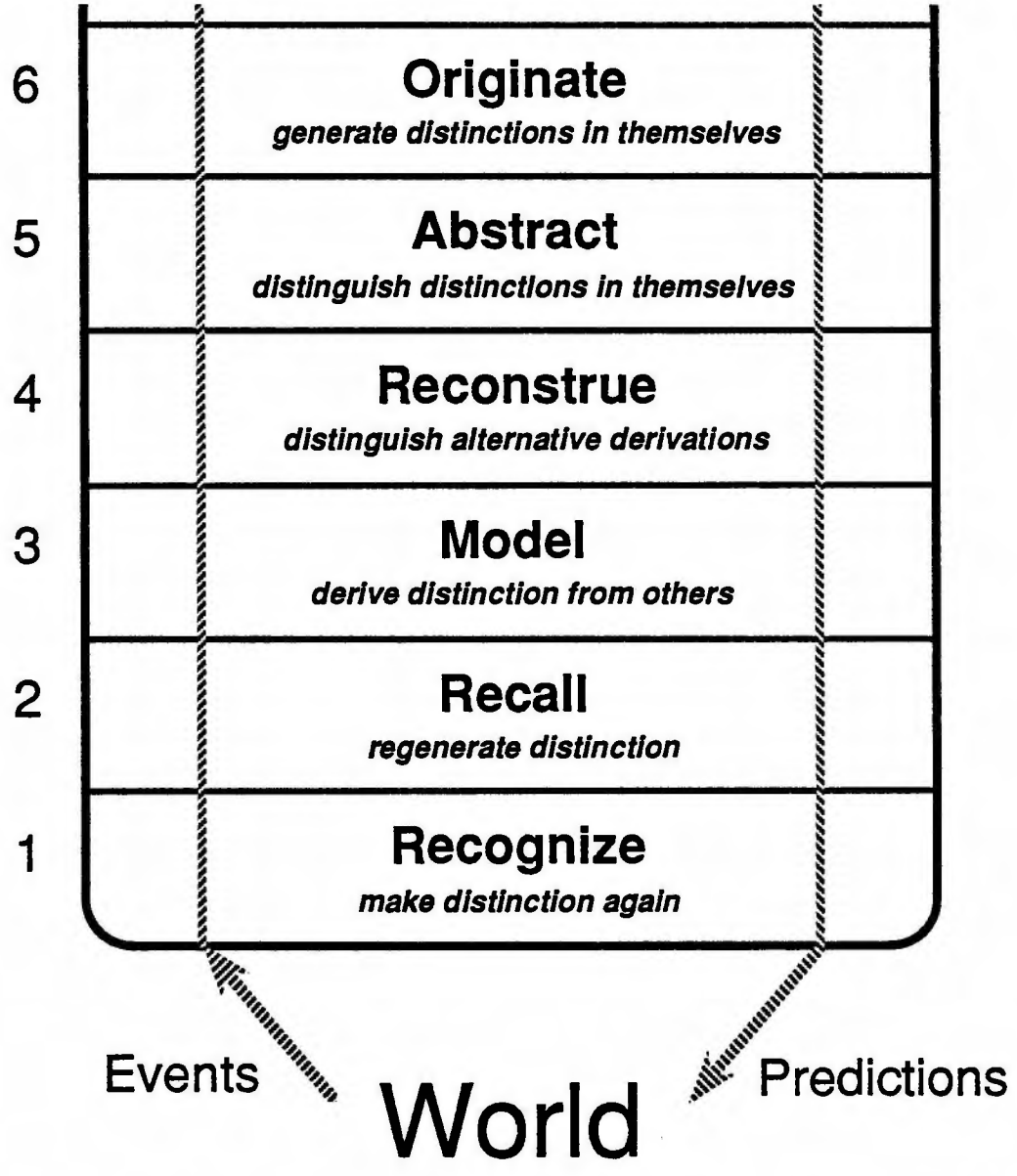


Fig.4 Anticipation as a hierarchy of distinction-making processes

- To *abstract* at the next level is the capability to detach distinctions from their sources and make the relations between models themselves subject to study. This facility to study the world of modeling as if it were a world of experience is fundamental to the externalization and growth of human knowledge as a cumulative by-product of the anticipatory process. It makes the expertise of the species largely independent of that of existing individuals.
- To *originate* at the next level is the capability to treat the distinction making process itself as a human activity, subject to choice and change, and to generate distinctions in themselves. This freedom raises many questions as to the nature of 'reality', of the 'wisdom' of certain distinctions, and of the relationship of the distinction-making, cognitive and knowledge processes to the nature of the individual and the survival of the species.

The anticipatory process encompasses interaction as much as it does modeling—we can anticipate the world by predicting where it is going or by controlling it to go a way with which we are equipped to deal. The combined hierarchies along the dimensions of increasing knowledge and increasing interaction give an operational model of power in the normal sense of the word, as the potential to change the course of events. Figure 5 shows the hierarchy of Figure 4 paralleled to encompass interaction with the world and used as a model of power.

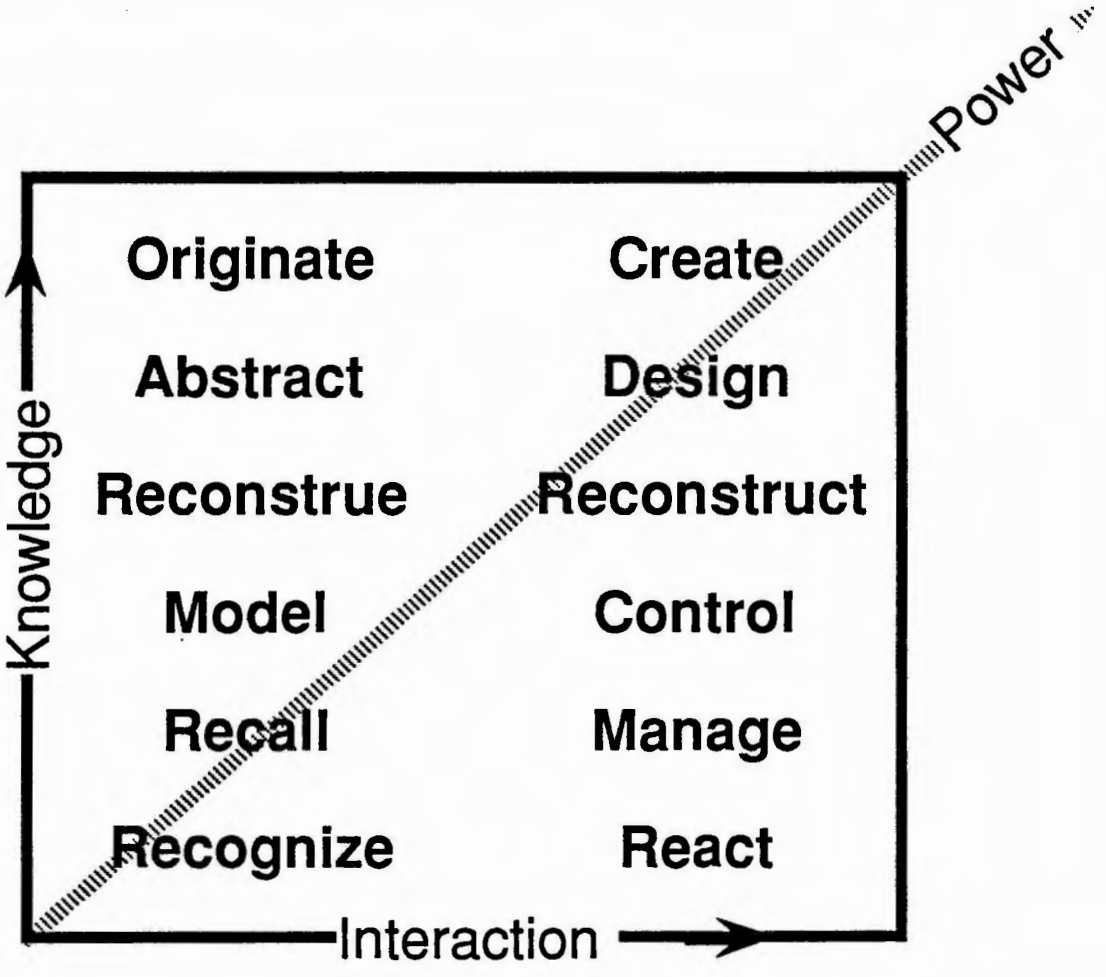


Fig.5 Interactive anticipation as a hierarchy of distinction-making processes

In terms of the interaction hierarchy:

- To *react* at the lowest level is the capability to respond to the same event again in the same way.
- To *manage* at the next level is the capability to regenerate reactions and use them outside their original context. This capability of managing situations has been noted by Hayes-Roth (1984) to be particularly suited to knowledge-based system precisely because it does not involve the use of deep models.
- To *control* at the next level is the capability to derive reactions from models.
- To *reconstruct* at the next level is the capability to change the models on which reactions are based.
- To *design* at the next level is the capability to derive the models from abstract distinctions.
- To *create* at the next level is the capability to generate new models and reactions.

PHYSICAL, MENTAL AND KNOWLEDGE WORLDS

One of the most important distinctions made in modeling experience of the world is to separate the world of other people from the physical world. Other people are recognized by the individual as analogies of the self, themselves anticipating the world, modeling and controlling it in a way unlike causal physical systems. In addition other people act as role models, knowledge sources, and creators of new knowledge, in a way that makes the dimensions of interaction with the social world very different from those of the physical world. Figure 6 recognizes this separation and shows the person having communication with a *mental* world in which perception involves the comprehension of communication, and action involves the capability to generate comprehensible explanations. I have used the term 'mental' very deliberately instead of 'social' or 'cultural' world because it emphasizes the lack of separation between the mental world of the individual and the equivalent phenomenon in society. The distributed anticipatory system which is the species is best regarded as a unitary knowledge processing entity, with individual knowledge processes differentiated and independent from the whole to a very small extent compared with their integration with, and dependence upon, that whole.

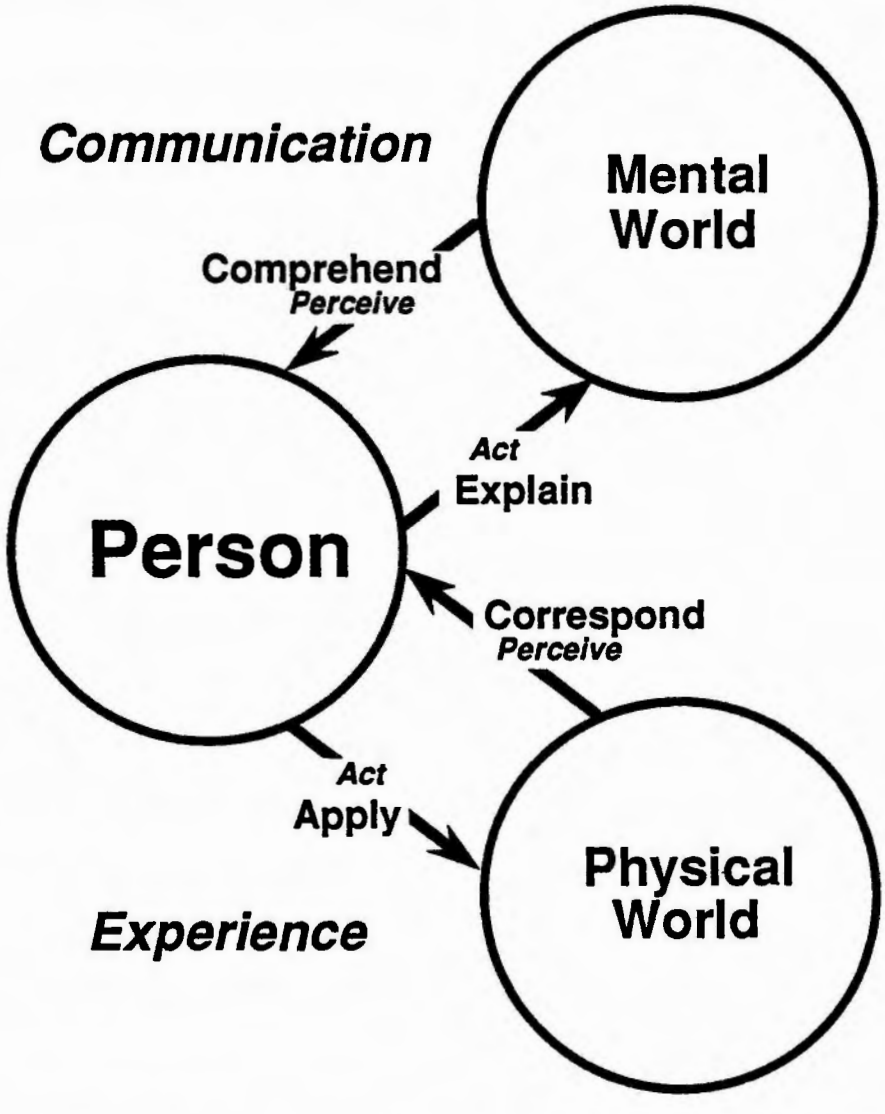


Fig.6 Person as an anticipatory system communicating with the mental world

Figure 6 also emphasizes the perceptions of and actions upon the mental world are evaluated in a different way from those relating to the physical world. It is possible to collapse the mental into the physical and treat the mental processes of people as if they were the physical processes of objects, attempting to predict and control them. Habermas (1984) terms this 'strategic' social action, involving including the decisions and behaviors of other people in a means-end calculation. Social actors in the strategic mode relate to other persons as objectifiable means or obstacles to the attainment of their aims. However, the mental world is one of human choice within a framework of established conventions and hence different modes of interaction with it are possible. It is, for example, important to communicate to others what family of conventions one intends to adhere to. Habermas terms this 'dramaturgical' social action involving the projection of a public image. It is also important to operate together as a social unit within a common framework of conventions. Habermas terms this 'normative' social action in which the parties involved fulfil reciprocal expectations by conforming their behavior to shared norms and values.

Habermas' fourth category of social action, *communicative action*, is particularly interesting because it involves the persons involved attempting to reach an explicit agreement on their situation for the sake of cooperation. This introduces the concept of mutual understanding and hence of mutual knowledge suggesting a further distinction that separates the shareable distinctions involved in communicative action from the less shareable distinctions involved in individual activities and social interaction without mutual understanding. Figure 7 extends Figure 6 to make this further distinction, showing the person interacting with a 'knowledge world' also, in which perception involves the derivation of knowledge from other knowledge, and action involves the capability to create new knowledge.

This knowledge world has many of the properties of Popper's (1968) world 3 of *objective knowledge*. In particular, it is a product of the human mental world, but distinguished just to the extent that it stands independent of the particular mental worlds from which it was created. There is an important conceptual link between Popper's criteria for objective knowledge and the existence of a social world valuing communicative action. The primary criterion is one of *fallibilism*—there must be tests that might show the knowledge to be false. However, this criterion which establishes the possibility of a critical community, also establishes the need for knowledge to be *understandable* so that it can be tested by the community (Notturno 1985).

Pask (1980) has developed a similar framework to Habermas as a foundation for computer-based learning. His *conversation theory* singles out as the critical event in discourse the occasions on which two participants agree that they have effectively exchanged a concept and validated its mutual similarity. Such *agreement over an understanding* punctuates the discourse into *strict conversations* whose structure and processes can be analyzed in depth as coherent units. Pask recognizes the diversity of validity tests that can be applied to understanding such as the ability to: *produce* knowledge transferred; to *reproduce* knowledge transferred from an alternative derivation; to *apply* knowledge transferred to problem solving; and, recursively, to *teach back* knowledge transferred. These criteria correspond to activities at different levels in Figures 4 and 5, instantiating the means open to people to undertake Habermas' communicative action through attempting to reach Pask's agreement over an understanding using Popper 's objective knowledge.

Figure 7 also shows some of the ways in which activities at different levels of the hierarchies of Figures 4 and 5 are supported through the social world. The reflexive capability to react to the physical world is paralleled by a corresponding capability to *mimic* in the social world. This is an important knowledge transfer mechanism that is supported by social action in exemplifying experience, communication and reasoning. The reinforcement processes in the physical world supporting the development of second level capabilities to recall and manage based on evaluated experience are extended by social reinforcement processes. The derivation of relations between distinctions at the third level of modeling and control is supported by explicit tutoring discussing experience, communication and reasoning.

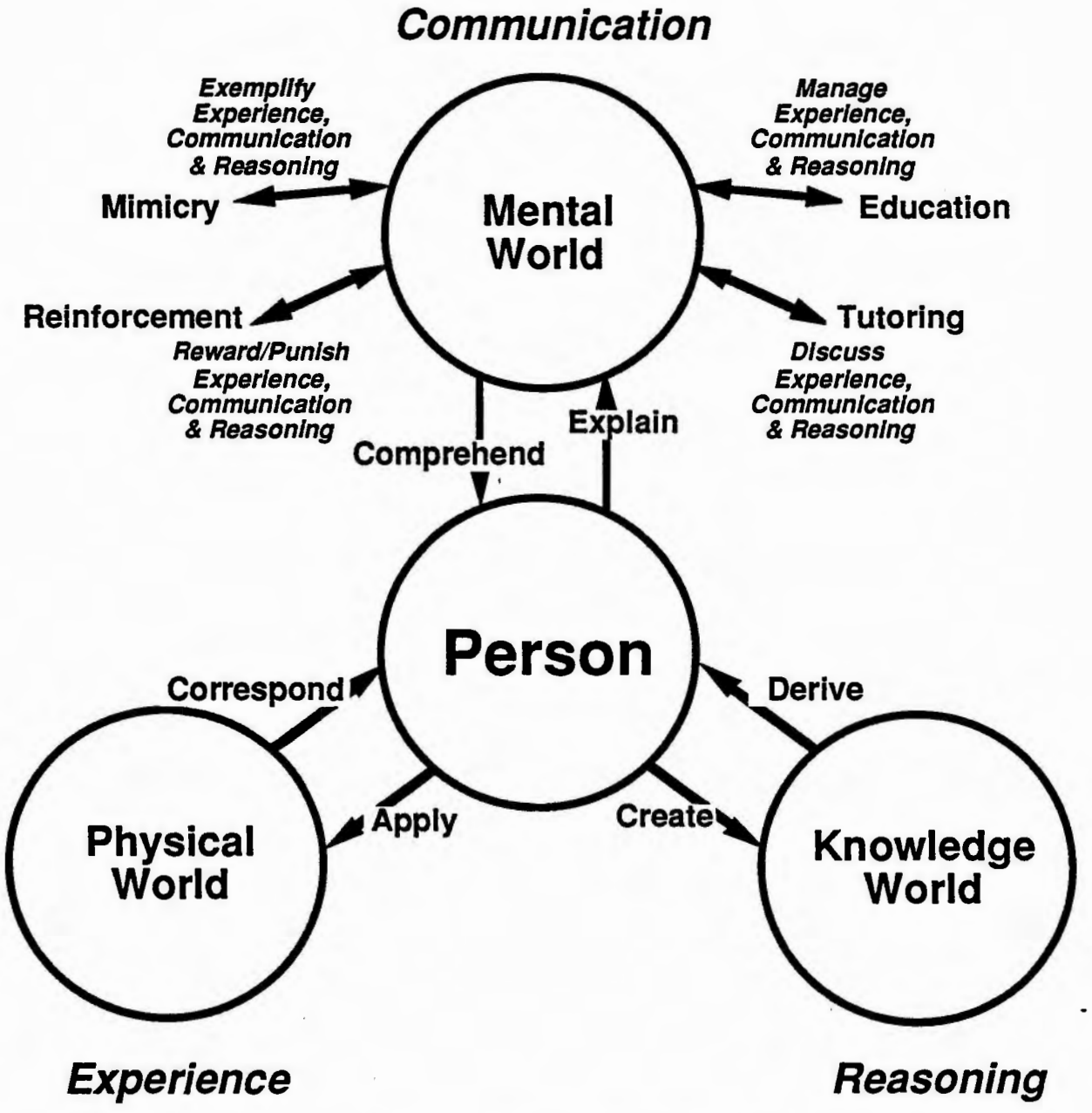


Fig.7 Person as an anticipatory system reasoning in the knowledge world

It is important to note also that all these processes may be applied recursively to enhance their own operations. We can mimic and reinforce those who are effective as exemplifiers, reinforcers and tutors. In particular, as indicated in Figure 7, we can manage the process of knowledge transfer through the educational system, although our knowledge of the acquisition process is such that we cannot yet go beyond this to control, reconstruction, design and creation.

THE ROLE OF KNOWLEDGE-BASED SYSTEMS

Figures 5 and 7 have two important inter-relations. The development of the distinctions between the physical, mental and knowledge worlds shown in Figures 3, 6 and 7, may be seen as a necessary component of the ontogenesis of the hierarchies of Figures 4 and 5. From an individual perspective, the person alone cannot develop this hierarchy without the support of the social infrastructure of other people, and hence the shared mental world is an important construct to be identified as separate from the physical world. We can, and do, of course participate within the socio-cultural world without explicitly modeling it and its effect on us, but the full freedom of the upper levels of the hierarchy involves the recursive application and extension of our models to include the routes by which we arrived there. The separation of the knowledge world is a consequence of our modeling the cultural mental world as separated from our individual mental world and residing in the minds of other individuals, that is, of imposing our models of our selves upon the knowledge processes of society. It corresponds to the detachment of distinctions from experience at levels 3 and above in the hierarchies of Figures 4 and 5.

From a distributed anticipatory system perspective, the separation of the individual and cultural mental worlds is an artefact of the communications interfaces and the limits of individual rationality. Conventional views of the individual mental world and the knowledge world result from our individual-centered perspective on the survival and evolutionary processes of the species. If we see the species as a the unit then the cultural mental world is the manifestation of increasing power and the knowledge world is a product supporting that power.

This is important in understanding and evaluating the role of computer technology in supporting human knowledge processes. Computer technology has largely supported the more abstract human knowledge processes in the past such as those involving modeling, mathematics and calculation. Expert systems may be seen as an attempt to support lower level processes where models have not been available and the tasks have had to be 'managed' by people. Robotics may be seen as as attempt to support even lower level processes of interaction with the world.

This move towards technological support of the lower levels when the growth of knowledge emphasizes the upper levels may appear surprising. However, it is not so. First, the primary pattern of development in our civilization has been to the release of human labor from the physical necessities of survival to the freedom and creativity of experiental, social and intellectual activities. The mechanization of agriculture and the automation of factories are major developments that are now being paralleled by the computer support of low-level mental and physical functions. Second, the development of the upper levels of the anticipatory hierarchy through inductive processes requires an increasingly massive flow of experience through the lower levels. The *grounding* of our knowledge in experience requires a wealth and diversity of experience, and a level of flow of communications, that the biomass alone can no longer encompass. Expert systems are one further step towards the enhancement of human experience through instrumentation and interaction technologies.

In these terms, knowledge-based systems in general may be distinguished from expert systems in particular through their integration of knowledge processes at all levels of the hierarchy. The term 'expert systems' arose because of the emphasis on a new paradigm of modeling the human operator managing those decision and control situations which are not amenable to modeling. That is, expert systems developments have focused precisely on those situations where our knowledge is only at level 2 in the hierarchies of Figures 4 and 5. Thus, it is not surprising that problems of knowledge transfer from human experts arise in the development of expert systems—these are the same problems that prevent the expertise involved rising from recognition and recall to modeling, from reaction and management to control.

In terms of these considerations, we should beware of two extremes. First, of assuming that all human expertise is of this nature—it encompasses the complete hierarchy of Figures 4 and 5. Second, of treating expert systems developments at level 2 as of lesser significance than the upper, more 'scientific', levels of the hierarchy. To the contrary, these developments may be viewed as part of a movement towards the major advances in the flow of information between levels 2 and 3 of the hierarchy, prepatory to similar advances between levels 1 and 2. Moving along the 'power' dimension of Figure 5 involves a very high level of activity at the lower levels if that at the upper levels is to be grounded in reality and applicable to the physical world.

THE EVALUATION OF KNOWLEDGE-BASED SYSTEMS IN 3 WORLDS

One immediate application of the concepts developed in Figure 7 is to the evaluation of knowledge-based systems, an important topic which is being increasingly addressed by empirical studies (Shaw and Woodward 1987) and the development of conceptual frameworks (Benbasat & Dhaliwal1988). In expert system developments, one naturally thinks in terms of validation against the expert, either against his or her knowledge explicitly expressed, or against his or her behavior. However, these 'subjective' dimensions of validation omit more conventional dimensions, such as: the evaluation of the overall performance of the system in applications; the correspondence of the knowledge in the system to empirical data; the derivation of the knowledge from established scientific knowledge; and the comprehensibity of the knowledge to the professional community as opposed to the individual experts.

Figure 8 shows the person in Figure 7 replaced by a knowledge-based system, together with links to the validation criteria used in expert systems and positive science, and with the 'worlds' filled in more formally in terms of relevant concepts and definitions. This diagram gives a framework allowing all six of the dimensions of validation mentioned above to be clearly defined, together with three others that may be less obvious.

The interactions of the system with some actual world in which it is intended to perform decision or control tasks introduce two dimensions:

- *Correspondence*—to what extent does the knowledge in the system correspond to actuality?
- *Application*—to what extend does the system perform the intended task?

These are major dimensions in conventional systems analysis and in the evaluation of a scientific or technological system. The assumption in expert systems development is that these may be difficult, or impossible, criteria to apply because of the nature of the problem. For knowledge-based systems in general, however, these conventional standards are still very important.

The interactions of the system with some mental world, for example that of the clients, for whom it is intended to perform decision or control tasks introduce two dimensions:

- *Comprehension*—to what extent is the system able to understand information and queries?
- *Explanation*—to what extend is the system able to explain its activities?

These are important dimensions in any system supporting human activities. There have been attempts to satisfy them in most knowledge-based system development from MYCIN/TEIRESIAS onwards. Conventional data-processing has often been weak in these areas.

The interactions of the system with some knowledge world underlying its performance of decision or control tasks introduce two dimensions:

- *Derivation*—to what extent can the system derive its required knowledge from well-established sources?
- *Creation*—to what extend is the system able to generate new knowledge?

The rational reconstruction of knowledge to show, regardless of its source, how it could have been derived from well-established sources is important in expert systems development. We would prefer the minimal content to be dependent on the authority of experts as opposed to their use of professional knowledge. Similarly, while the creation of new knowledge is not the primary focus of system development, it is one of the most valued achievements—"we now think about the problem in different terms" is a significant positive comment resulting from a knowledge-based system development—it would be even more significant if it were a result of system activity.

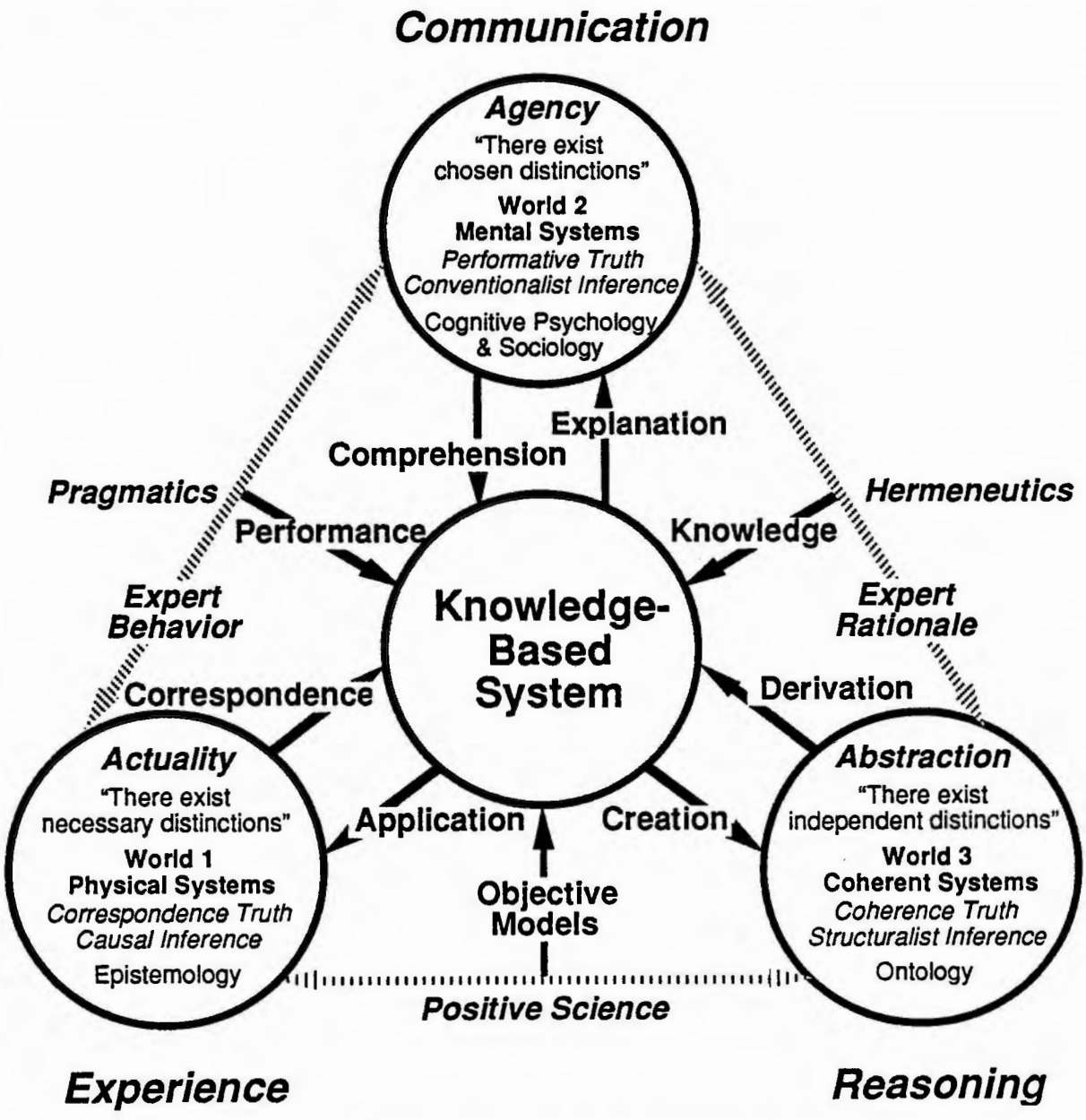


Fig.8 Dimensions of evaluation of a knowledge-based system

There are three ancillary dimensions in Figure 8 relating to interactions between the worlds. Expert performance corresponds to a relation between world 2 and one of the others—the link shown is to world 1. We can evaluate the behavior of the system against the behavior of experts. This is a pragmatic evaluation based on the known utility of that expert behavior.

Expert knowledge corresponds to a relation between world 2 and world 3. We can evaluate the rationale of the system against the rationale of experts. This is a hermeneutic evaluation based on respect for the conceptual framework in which the experts perceive themselves as operating.

The objective knowledge of positive science is a relation between world 1 and world 2 that combines correspondence, derivation and explanation. It is usually not applicable in the situations for which expert systems are being developed. However, the concepts and techniques underlying knowledge-based systems in general should have a seamless interface to this highly refined form of knowledge.

STRUCTURES OF KNOWLEDGE AND THEIR TECHNOLOGICAL SUPPORT

Knowledge-based systems have not developed in a vacuum. They are part of the evolution of computer technology and reflect trends in that technology towards end-user computing including end-user system development and maintenance. Knowledge is not something remote from human experience. It is closer to it than many of the technological devices that were necessary to bring computer technology into use in the early years of data processing. The use of knowledge, overtly and naturally expressed, to 'program' a computer system to provide well-presented and explained advice, is a natural step in computer product evolution.

There are many parallel paths converging on knowledge-based, user-oriented systems. *Hypertext* and *hypermedia* systems approach knowledge a different direction to that of expert systems, accepting it as having limited structure and processibility, and adding a limited structure through the links that is intended to aid the user rather than allow computer-based inferencing (Conklin 1986, Gaines & Vickers 1988). They extend knowledge representation to include any media whereby human knowledge can be transferred even if there is no means for analyzing that knowledge or its transfer. For example, a film of a skilled person performing a task which enables another person to perform that task better contains knowledge which can be accessed and transferred through a hypermedia system (Vickers & Kingston 1987).

Computer-aided software engineering (CASE), although the terminology emphasizes existing technological concepts, is itself a major step towards simplifying the application of computer systems (Martin & McClure 1988), and many of the concepts and techniques involved closely resemble those being developed for knowledge acquisition for knowledge-based systems (Gaines 1988c). A wide range of developments in computing, in scientific visualization, in animated simulation, in document preparation, in symbolic mathematics, in natural language interaction—all can be seen as *knowledge support* systems for knowledge-based human activities. In knowledge acquisition for knowledge-based systems we are often concerned to integrate these other developments. Do the models developed in this paper provide a framework for this integration?

Figure 9 analyzes major intellectual developments that are subject to technological support focusing on the path from human experience to the most refined scientific abstraction, currently category theory. It shows critical distinctions at each stage, for example, that between passive observation and the active, intentional processing of information that is peculiar to living systems. It shows the distinctions of Figures 4 and 5 leading through anticipation to modeling, and through interaction to control. It shows how experience is shared through communication allowing anticipation to be an activity of the species rather than of the individual. It shows how communication is refined

through evocation of desired responses in rhetoric, accurate description of experience through representation, and the structured argument of discourse. It shows that the application of rationality criteria to discourse leads to the synthesis of new knowledge through induction and the analytic development of existing knowledge through deduction, arithmetic, logic, algebra and category theory.

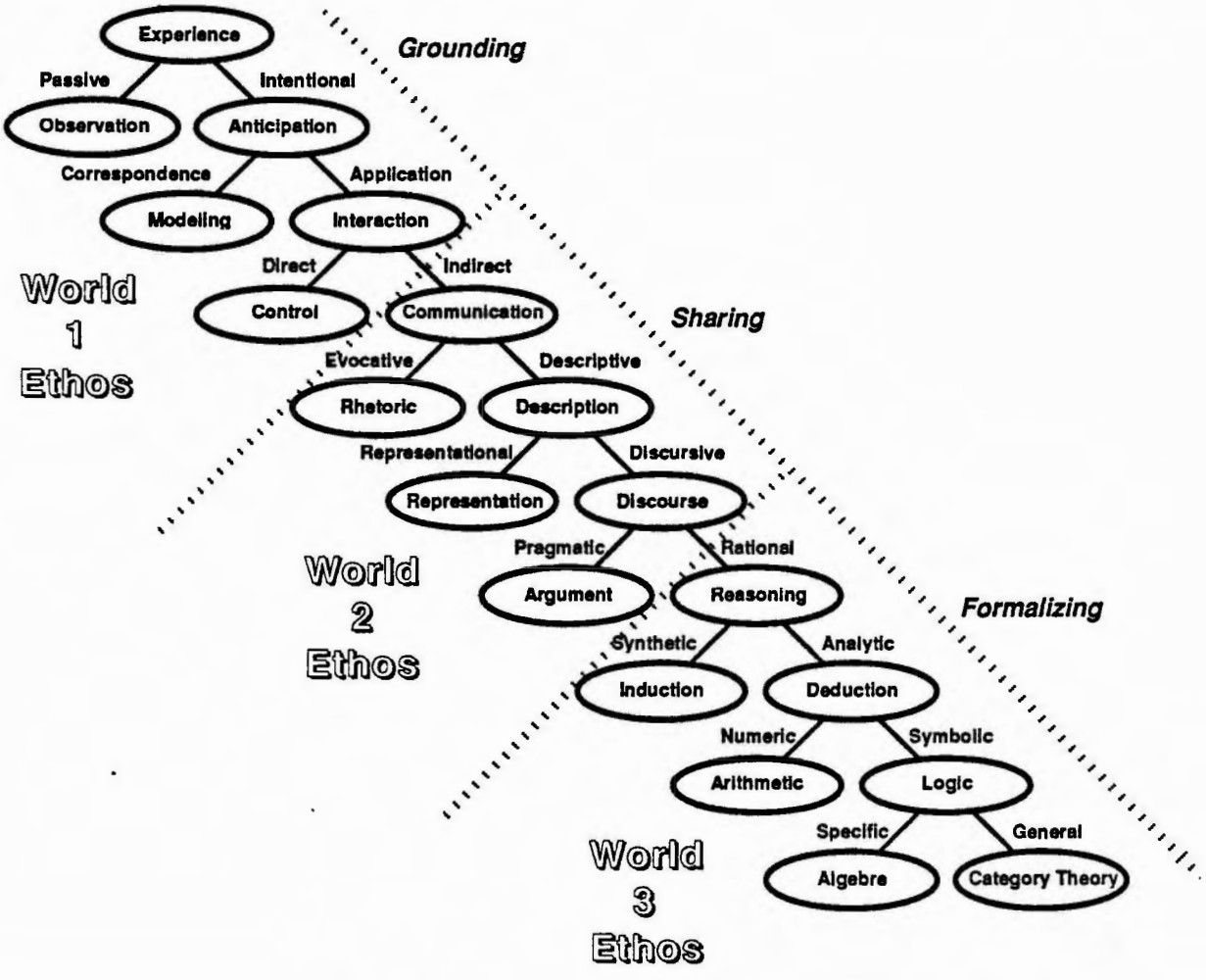


Fig.9 Human intellectual developments

Figure 9 presents a level of detail below that Figure 7, marking some of the major intellectual developments of the human species in constructing an inductive hierarchy of knowledge. The table in Figure 10 sets out major knowledge processes from Figure 9, the relevant techniques and supporting technologies, pre- and post- the development of computing. Looking down the right hand column of Figure 10 we can see the full spectrum of computer-based techniques supporting human knowledge processes. It is interesting to note the way in which the development of computing has straddled the central region of discourse and inductive reasoning. Computing commenced with number crunchers, moved into simulation and instrumentation, and then into databases, graphics, text processing. The support for the explanatory processes of discourse and the argument forms of practical reasoning are still very weak, even though these underlie our colloquial use of the term 'expertise.' Current expert systems largely support deductive reasoning and may be seen as a natural extension of database technology to allow retrieval of logically derived fields.

Human Knowledge Processes	*Relevant Techniques*	*Supporting Systems*	
		Pre-Computing	*Post-Computing*
Experience	Perception, Memory	Instruments, Data Recording	Instrumentation, Data Storage
Anticipation	Cognition, Modeling	Modeling Techniques, Statistics	Data Analysis
Interaction	Action, Control	Tools, Mechanisms	Simulation, Robotics
Communication	Rhetoric, Mimicry	Pamphlets, Novels, Drama, Movies	Games
Description	Representation, Language	Representational Painting & Writing, Phonograph, Television	Word Processing, Graphics, Information Retrieval
Discourse	Argument, Communicative Action	Books, Journals, Journalism, Telephone, Education System	Electronic Mail, Hypermedia, CAL, Natural Language
Reasoning	Induction, Abduction, Analogy	Jurisprudence Systems, Uncodified 'Commonsense'	Semantic Nets, Conceptual Graphs, 'Advice Systems'
Deduction	Mathematics Set Theory, Arithmetic	Calculators, Diagramming Techniques	Number Crunchers, Relational Databases, 'Expert Systems'
Logic	Propositional & Predicate Calculi, Modal Logics	Proof Techniques, Sequent Calculi	Algebraic Manipulators, Theorem Provers
Category Theory	Universal Algebras, Topology, Homology	Metatheorems, Proof Techniques	Category-Theoretic Programming Languages

Fig.10 Human Knowledge Processes, Techniques and Supporting Systems

I have shown 'advice systems' as a place marker for what might be expected to be the dominant role of an 'expert system' emulating the main activities of a human expert, the knowledge organization and creation activities of inductive, abductive and analogical reasoning that are an integral part of expertise in the model developed in this paper. These activities are treated as 'updating' and 'maintenance' in current expert system technologies, but little provision is made in the deductively oriented shells to support these activities through techniques discussed earlier such as anomaly detection and reporting. The current knowledge acquisition literature focuses on rapid prototyping tools for knowledge engineering rather than on the integration of acquisition activity with the ongoing application of the knowledge-based systems. This is reasonable at the current state of the art, but integration of acquisition and performance will become a major design issue as knowledge-based systems come into wider use and the problems and expense of 'updating' and 'maintenance' become apparent.

The need for integration of computing technologies is very apparent in Figure 10. Practical knowledge support systems must be capable of providing combinations of graphics, text processing, hypermedia, number crunching, databases, communications, theorem proving, databases, and semantic nets and expert system technologies. These are not highly separated functions but rather multiple aspects of human knowledge processing.

CONCLUSIONS

A model of knowledge-acquisition for knowledge-based systems has been presented which presents the acquisition activity as playing an essential and continuous role in skilled performance, rather than as a separate and separable activity. The practical implications of this model for systems design have been developed, and recommendations made targeted on monitoring the quality of advice from expert systems and achieving closer integration between the application of these systems and the formation of expertise. The model has been developed in depth to generate taxonomies of human knowledge processes and use these to analyze the roles of a wide variety of computer-based systems in supporting these processes. The model has been used to highlight strengths and weaknesses in the current state of the art in knowledge-based systems. An overall framework has been provided for the variety of knowledge acquisition problems, techniques and technologies discussed in the literature.

ACKNOWLEDGEMENTS

I am grateful to many colleagues for discussions over the years that have influenced this paper. In particular I would like to thank John Boose, Jeff Bradshaw, Bruce Conrad, George Klir, Debbie Leishman, Gordon Pask, Mildred Shaw, Joan Vickers and Ian Witten, for access to their own research and many stimulating discussions. Much of the research reported in this paper is part of a long-term collaborative study with Mildred Shaw of the cognitive foundations of human knowledge processes.

REFERENCES

Benbasat, I. & Dhaliwal, J.S. (1988). A framework for the validation of knowledge acquisition. **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff (November).

Blume, S.S. (1974). **Towards a Political Sociology of Science.** New York: Free Press.

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.

Conklin, J. (1986). A survey of hypertext. MCC Technical Report STP-356-86. Austin:Microelectronics & Computer Technology Corporation (October).

Davis, R. & Lenat, D.B. (1982). **Knowledge-Based Systems in Artificial Intelligence.** New York: McGraw-Hill.

Feyerabend, P. (1975). **Against Method.** London: NLB.

Gaines, B.R. (1987a) An overview of knowledge acquisition and transfer. **International Journal of Man-Machine Studies 26**(4), 453-472 (April).

Gaines, B.R. (1987b). How do experts acquire expertise? **Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems.** pp.B2-1-11. UK: Reading University (September).

Gaines, B.R. (1987) Advanced expert system support environments. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.8-0-8-14. Banff (October).

Gaines, B.R. (1988a). Second generation knowledge acquisition systems. **Proceedings of the Second European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'88).** GMD-Studien Nr.143. pp.17-1-17-14. Bonn, Germany: Gellsellschaft für Mathematik und Datenverarbeitung (June).

Gaines, B.R. (1988b). Positive feedback processes underlying the formation of expertise. **IEEE Transactions on Systems, Man & Cybernetics,** to appear.

Gaines, B.R. (1988c). Software engineering for knowledge-based systems. **Proceedings of CASE'88, Second International Workshop on Computer-Aided Software Engineering.** 14-1-14-7.

Gaines, B.R. & Vickers, J.N. (1988). Design considerations for hypermedia systems. **Microcomputers for Information Management,** 5(1), 1-27 (March).

Habermas, J. (1981). **The Theory of Communicative Action: Reason and the Rationalization of Society.** Boston: Beacon Press.

Hagstrom, W.O. (1965). **The Scientific Community.** New York: Basic Books.

Hawkins, D. (1983). An analysis of expert thinking. **International Journal of Man-Machine Studies, 18**(1), 1-47 (January).

Hayes-Roth, F. (1984). The industrialization of knowledge engineering. Reitman, W., Ed. **Artificial Intelligence Applications for Business.** pp. 159-177. Norwood, New Jersey: Ablex.

Klir, G.J. (1985). **Architecture of Systems Problem Solving.** New York: Plenum Press.

Martin, J. & McClure, C. (1988). **Structured Techniques: The Basis for CASE.** Englewood Cliffs, New Jersey: Prentice Hall.

Merton, R.K. (1973). **The Sociology of Science: Theoretical and Empirical Investigations.** Chicago: University Press.

Notturno, M.A. (1985). **Objectivity, Rationality and the Third Realm: Justification and the Grounds of Psychologism: A Study of Frege and Popper.** Dordrecht, Holland: Martinus Nijhoff.

Pask, G. (1980). Developments in conversation theory—Part I. **International Journal of Man-Machine Studies, 13**(4), 357-411 (November).

Popper, K.R. (1968). Epistemology without a knowing subject. Van Rootselaar, B., Ed. **Logic, Methodology and Philosophy of Science III.** pp. 333-373. Amsterdam, Holland: North-Holland Publishing Co.

Rappaport, A. & Gaines, B.R. (1988). Integration of acquisition and performance systems. **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff (November).

Schön, D.A. (1983). **The Reflective Practitioner.** New York: Basic Books.

Shaw, M.L.G. & Gaines, B.R. (1988). A methodology for recognizing consensus, correspondence, conflict and contrast in a knowledge acquisition system. **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff (November).

Shaw, M.L.G. & Woodward, J.B. (1987) Validation of a knowledge support system. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.18-0-18-15. Banff (October).

Stich, S.P. & Nisbett, R.E. (1984). Expertise, justification and the psychology of inductive reasoning. Haskell, T.L., Ed. **The Authority of Experts**. pp. 226-241. Bloomington, Indiana: Indiana University Press.

Vickers, J.N. & Kingston, G.E. (1987). Modelling the master coach: building an expert system for coaching. Norrie, D., Ed. **Learning in Future Education: Proceedings of International Conference on Computer Assisted Learning in Post-Secondary Education**. pp.207-212. University of Calgary.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

An Interactive Performance Based Knowledge Acquisition Methodology

William A. Gale

AT&T Bell Laboratories
Room 2C278
600 Mountain Avenue
Murray Hill, NJ, 07974
USA

ABSTRACT

Expert systems are systems which model an expert and which achieve expert level performance. Many systems demonstrating feasibility of using this approach to constructing consultation systems have been built. However, the systems are massive and hard to build, so that it is difficult to study how they should be built.

Microstrategies are formal models of how simple decisions are made by one person. The decisions modeled are simple in that they have just a few possible alternatives. The resulting models are simple analogues of the the larger models needed for consultation systems. They are simple to build, taking a day or less of an expert's time, so it is possible to build a number of models to study. Still, the resulting models typically show interesting structure in the form of several distinct reasons entering the decisions, so lessons learned by studying these models may be useful in building larger models of experts. In particular, the models open the possibility of research on how to combine (structured) models from several disagreeing experts.

There are other reasons for building microstrategies. The models built may have a use, even though they only represent the decision making of just one person, for instance as a default in some calculation or display. Also, building models of how various experts make the decision in question is a way to begin to formalize a decision. It may be a useful viewpoint for preparing tutorial material.

1. Introduction: Microstrategies Model Simple Decisions

A microstrategy is a formal model of how a simple decision is made. This paper describes a methodology for acquiring microstrategies characterized by its interactivity and by being performance based. Currently, most of the models built with the method are in the statistics domain. The final section of the paper discusses how the methodology may be extended to other domains. An example of a microstrategy will illustrate the kind of decision modeled and the formal models built.

The decision modeled is whether to superpose two time series in one panel, or to juxtapose them in two adjacent panels, supposing that they are to be shown against the same scale. This is a decision that has not been formalized before. The examples used were selected by a model of whether to plot two time series in juxtaposed panels using the same scale, the same units per centimeter, or different scales. Figure 1 illustrates the decision.

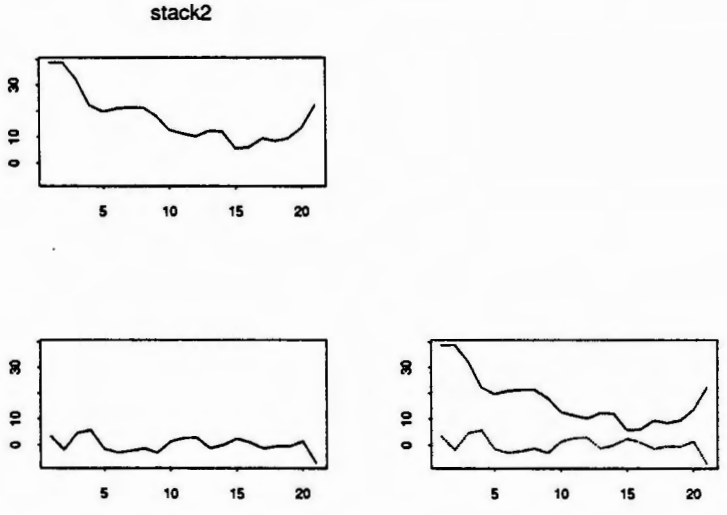


Figure 1. The two panels on the left juxtapose displays of two time series, while the single panel on the right superposes them. For this example, the model calls for superposition because the variables have no crossings.

The model is

If correlation between the two variables is not small [corr>(.47,.53)]
  Then select superposed in one panel
    (selects 30 out of 81)
If log of the number of crossings of the series is not large [crsal<(1.4,1.1)]
  Then select superposed in one panel
    (selects 24 out of 51)
If log of the lesser average slope is small [rsl<(.69,-1.3)]
  Then select superposed in one panel
    (selects 6 out of 27)
If difference between the means is not small [mdiff>(.23,.37)]
  Then select superposed in one panel
    (selects 3 out of 21)
Otherwise select juxtaposed in two panels
  (selects all the remaining 18 cases.)

The model consists of a series of "rules," each giving a hypothesis, and the decision if the hypothesis is satisfied. The English versions of the hypotheses give a qualitative summary of the precise versions suggested in square brackets. The qualitative form of these rules is similar to advice that can be found in statistics books, and might be called an informal model. That form is vague, but can give people enough guidance to be able to think about a specific case.

The precise formulation of each rule specifies a statistic and limits to be used to interpret the statistic. This set of rules is *formal* because it specifies the decisions objectively and operationally. The result can be calculated for any two series of equal length.

The definition of the statistics cited by these rules is an integral part of the formal model. In this model, "corr" is the Pearson product moment correlation. "crsal" is the logarithm of one plus the number of crossings of the two series. "mdiff" is the difference between the means of the two series divided by the range of the data in both series. "rsl" is the log of the lesser of the average absolute slopes. That is, denoting the two series by x_i, y_i, i=1, $\cdots$,N, their ranges by $r_x = \max_i(x_i) - \min_i(x_i)$ and r_y, and the physical extents of the data region horizontally and vertically by r_h and r_v, then the slope of one segment of the x series is

$$\frac{\Delta v}{\Delta h} = \frac{\dfrac{r_v}{\max(r_x,r_y)} \mid \Delta x_i \mid}{r_h/N}$$

Thus the average absolute slopes are

$$b_x = \frac{Nr_v}{(N-1)r_h} \frac{\sum_{i=2}^{N} \mid x_i - x_{i-1} \mid}{\max(r_x,r_y)}$$

and

$$b_y = \frac{Nr_v}{(N-1)r_h} \frac{\sum_{i=2}^{N} \mid y_i - y_{i-1} \mid}{\max(r_x,r_y)}$$

The statistic, "rsl," is $\min(\log(b_x),\log(b_y))$.

The model also specifies two limits for interpreting the statistics, a "loose" limit, and a "tight" limit. In the display above, the loose limit is given first, the tight limit second. When a statistic satisfies the inequality using the tight limit, the rule will always be applied. When a statistic does not satisfy the inequality using the loose limit, the rule will not be applied. The region in between is a "grey" region in which the rule may or may not be applied.

The formal model of superposition is an example of a model of a single simple decision. Although I'm unclear as just what constitutes a "simple" decision, three aspects are worth noting. The decision has only two alternatives; it can be made from a simple input, two equal length series of real numbers; and I made about 80 of these decisions in less than an hour. The last of these three observations seems to me to characterize a "simple" decision, but the first two have been also true of all the models I have built. Given a simple decision, it is reasonable that the formal model that results is also simple.

The process to generate such simple models is also simple compared to the process for generating a model of the size of REX [Gale, 1986]. I built the model above in a week, compared to two to three person-years required to build REX. Since developing some software to facilitate making models similar to the one above, the time to make a model has dropped to less than a day.

2. Data Based Decision Model Building

We begin the process of model building with a person and a decision. The decision is one with just a few alternatives, and concerns the treatment of some data. The person is able to gather example data sets and to make the decision given a specific data set. The person does not necessarily know how the decision is made. The following steps have proved useful in building formal models of the decision in this setting.

1. generate examples
2. define the display upon which to base judgements
3. make judgements *
4. define possibly discriminating statistics
5. discriminate judgements *
6. review judgements *
7. generate rules *

* supported by computer software

After defining the decision alternatives, the first three steps generate instances of the decision to be studied. The first step is to produce examples of the inputs to the decision. One hundred examples seems to be manageably few, yet sufficiently numerous to give credible results. If a set of examples is not already available, this step may take several days. However, many decisions can reuse data sets collected for previously studied decisions.

The second step defines what information will be used to make the decision. Since the decision is not formalized already, making the decision is probably not a matter of calculating some statistic, but of making some graphical presentation. A numerical or semi-graphical display, such as a stem-and-leaf diagram might also be used however. A few tries at an appropriate set of information may be needed.

The third step generates (preliminary) judgements to be analyzed. It usually takes an hour, or perhaps two, once a satisfactory information display is defined, to make all the judgements. It also makes the statistician aware of some of the qualitative bases of the decision. By making notes about the decisions as they are made, the statistician can begin to articulate why some of the decisions, at least, are made. This induces regularity into the judgements as well as providing a basis for beginning to study the judgements.

The last four steps are done repeatedly until a satisfactory model of the judgements is found. From the notes made while judging, the statistician usually has several possible statistics to

investigate. Programming and testing them was usually done in less than an hour in building the seven models summarized later. The statistics defined at this point may be common functions, such as a product moment correlation. More often, they are quantities tailored to the decision at hand, such as the number of crossings of two lines or the average slope of line segments in a graph.

The fifth step is the key step in formulating rules for the model. A plot, such as that shown in Figure 2, displays the judgements against one or two of the defined statistics. The alternative selected is shown by a letter, lower case if the judgement is unconfirmed, upper case if confirmed, or by two letters with an upper case letter for the preferred alternative and a lower case letter for an acceptable alternative. These plots show which examples are similar to others as judged by the statistics chosen for axes. If a judgement specifying one alternative is surrounded by judgements specifying another alternative, it is natural to reconsider the first judgement.

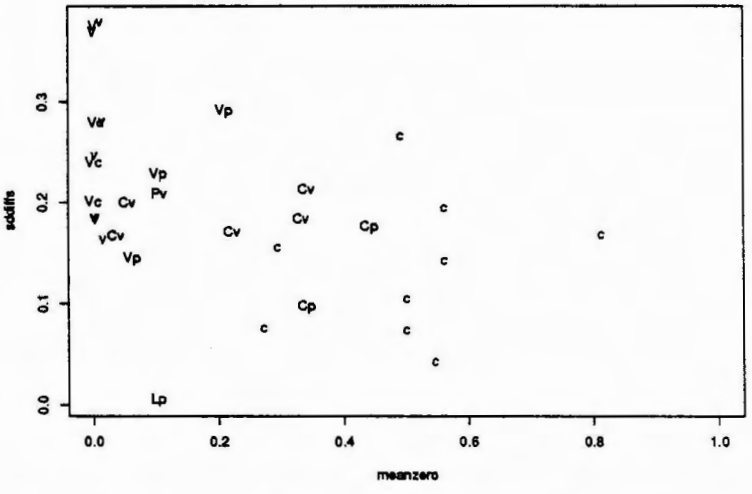


Figure 2. Judgements are displayed against two possible statistics. The alternative selected is shown by a letter, lower case if the judgement is unconfirmed, upper case if confirmed, or by two letters with an upper case letter for the preferred alternative and a lower case letter for an acceptable alternative. Possible rules select cases in half planes or rectangles including a corner, and can be found by inspection. A key function of the plots is to identify key examples worth reexamining carefully.

In reconsidering judgements, there have been two outcomes. The more frequent outcome is that the judgement is changed. However, an important outcome possible is that the statistician reconfirms the judgement. In this case it has been my experience that the statistician can also say *why* the example differs from those near it in the plot. If no currently defined statistic captures this reason, then the statistician defines another statistic to do so.

The geometrical representation of a rule in the judgement plot is an area. All examples in the area will be selected by the rule. I have used two specific kinds of rules. For the simpler one-hypothesis rule, the corresponding area is a half plane to the right or left, or to the top or

bottom. The other kind of rule I have used is a conjunction of two hypotheses (If hyp1 AND hyp2, then select ...). The corresponding area is a rectangle which includes exactly one corner in the plot.

When the judgements of a number of examples satisfy one of these patterns, then it is useful to define a rule. I usually seek a rule that will select a fifth or more of the remaining examples. In some cases, I have selected rules that selected only a few points. These rules were usually ones that I had anticipated in my notes, and that seemed to be required to remove exceptions to a rule selecting more examples.

2.1 Software to Support the Methodology

A software tool has been built that supports this way of building formal models in statistics. Three of the steps still need to be done by hand: generating the examples, defining the display from which to make the judgement, and defining the statistics. Given the examples and the display, then the software collects preliminary judgements and notes. The software presents the examples in a random order by default, although the input order or alphabetical order can also be used. Specific judgements can be reviewed, or all the examples judged as a specific alternative can be reviewed. Judgements can be changed, augmented, or reconfirmed. An augmented judgement specifies a preferred alternative and an acceptable secondary alternative. The notes collected while judging can be gathered, sorted by judgement, and printed out. Notes made while making preliminary judgements are a fruitful source of ideas for statistics.

After collecting a set of judgements, and with statistics defined for the examples, rule building begins. The key display is the judgement plot (such as Figure 2). By pointing at a specific judgement on the judgement plot, one can review the information display for the underlying example. If the judgement is changed, then the judgement plot will be updated immediately. Additional statistics can be defined using the full powers of the S language.

Rules are selected graphically, pointing to the side or corner which is to be included in the half plane or rectangle of the rule. The target decision of the rule is also shown graphically. The software then constructs the most inclusive rule of the kind requested that is consistent with the judgements recorded. The tight limit is set to include only examples for which the target decision is the preferred judgement, while the loose limit is set to include only examples for which the target decision is an allowed alternative. Examples with the target decision not included in the rule may be handled by another rule, or by revising some judgement. As rules are defined, additional plots of judgements show only the examples not selected by the loose limit of some rule.

The software supports exploring different models by maintaining a tree of considered models. One can return to any previously defined sequence of rules and consider continuations from that point. When an acceptable model is found, it can be compiled into a function that will execute rapidly. The tree of considered models allows work to be interrupted, saved, and resumed.

The software is written in S (Becker, Chambers and Wilks, 1988), which is available for any UNIX system. S is an interpreted interactive language designed for numerical and symbolic processing. A user manual for the knowledge acquisition software described here is available.

2.2 Formal Structure of the Models

The models being investigated have a sharply restricted form. The form can be summarized as follows:

strategy := (rule1, rule2, ..., target)
rule := (hypothesis, target)
rule := (hypothesis1, hypothesis2, target)
hypothesis := (statistic, direction, tight limit, loose limit)

That is, a strategy is a sequence of rules and a target. A target is one of the alternatives for the decision being modeled. The target associated with a strategy is the default, the alternative to be chosen if none of the rules is satisfied. The rules are examined in the sequence specified.

A rule has either one or two hypotheses and a target. If all of the hypotheses of a rule are satisfied, then the target of the rule becomes the alternative selected by the strategy. The rules are examined in the sequence specified by the strategy; only the first rule whose hypotheses are all satisfied will be used.

A hypothesis is a quadruple. The statistic is a function from the decision inputs to real numbers. The direction is either "greater than" or "less than." The limits are both real numbers. The hypothesis is satisfied if the statistic is in the direction-specified relation to the tight limit. The hypothesis is not satisfied if the statistic is not in the direction-specified relation to the lower limit. When the statistic lies between the two limits, the hypothesis may or may not be satisfied; this interval is called the "grey region."

The grey region allows various context dependent choices for the software implementing a microstrategy. If interaction with a user is allowed, the user can be queried. Otherwise, a global parameter ranging between zero and one can be used to interpolate between the loose and tight limits. If the statistic is in the direction-specified relation to the interpolated limit, then the hypothesis is taken as satisfied.

This class of models can be compared to decision tree models. A decision tree is a tree with rules at each internal node and an alternative at each leaf. CART [Breiman *et al.*, 1983] is an example of an automated procedure for deriving a specific kind of decision tree. The class of models specified here restricts the form of the model from a tree to a sequence. While allowing rules with a conjunctive hypothesis restores some expressivity, the models considered here are not as expressive as full tree based models. This difference leads to a difference in the acquisition of the models and to a corresponding difference in their intelligibility.

In a rule tree, the formal equivalent to the two hypothesis rule is a pair of rules each having one hypothesis. In rule tree development, the examples not selected by the formally equivalent pair of rules fall into two groups - those cases not selected by the first rule, and those selected by the first, but not by the second. These two groups of examples are then treated differently in further rule development. In the sequential models described here, all the examples not selected by a two hypothesis rule are treated together in the following model development. The practical effect is that for the tree model, the two rules implementing a conjunction must introduce two different following submodels, while the the single rule of the sequential model introduces a single following submodel. The tree model is therefore harder to understand, although it is easier to construct.

There are also two differences between how one *acquires* a decision tree model and how one acquires
a model of the form described here. The differences stem from using an automated process versus using an interactive process.

The ability to specify not only a preferred choice, but also an acceptable alternate, is one such difference. When more than one action is acceptable, being able to see that it is leads to a greater flexibility in building a model. This makes it a harder problem for which to find an

optimal solution, and thus harder for automatic acquisition. However, it is easier to find a heuristic solution, and thus easier for interactive acquisition.

Planning to handle judgement changes and additional statistics is intrinsically interactive. The judgement plots provide a natural guide to reevaluating decisions which is not provided by tree building software, since it shows anomalous examples having decisions different from those around them. Likewise, a notion of which additional statistic may be useful flows from reconfirming a judgement in an anomalous example. This guidance is not supported by automated tree building software.

To summarize, the form of the models discussed here leads to models which are easily understood, and easily acquired. The restrictions on expressiveness need to be examined empirically. So far, the indications are that for the simple decisions being studied, the form is sufficiently expressive.

3. Discussion of the Methodology

3.1 Treatment of Standard Problems

Gaines (1987) gives a list of problems cited in literature about accessing an expert's knowledge. It is useful to examine how the methodology described here treats each of these problems. The italicized texts are quotes from Gaines.

"Expertise may be fortuitous. Results obtained may be dependent on features of the situation which the expert is not controlling." This problem and a related one, that the expert may be controlling aspects of the situation without being aware of doing so, are very difficult to overcome. The methodology here makes two contributions. First, the 'grey' region allows deferring some of the decision until unquantifiable context is known by a user. The regions where a rule always or never applies should be less dependent on uncontrolled or over-controlled situations. Second, the selection of examples is the main way that situation will enter. If the examples can be taken from a stream of examples encountered in consulting, then they can be expected to be representative.

"Expertise may not be available to awareness. An expert may not be able to transmit the expertise by critiquing the performance of others because he is not able to evaluate it." This problem is solved by a performance based methodology. By definition, the expert can *do* the task at hand. Therefore, the expert is competent to make the decisions given the examples. The expertise does not need to be available to awareness initially, although it is quite likely that it will become available as the process is followed.

"Expertise may not be expressible in language. An expert may not be able to transmit the expertise because he is unable to express it." A methodology based on performance solves this problem if we interpret 'language' as 'natural language.' The expert does not need to speak about the task, only to do it. If we broaden 'language' to include 'knowledge representation,' then a fairly well defined technical problem remains: is the knowledge representation expressive enough for the decisions modeled. This issue is discussed in the next subsection.

"Expertise may not be understandable when expressed in language. An apprentice may not be able to understand the language in which the expertise is expressed." Since a machine interpretable function is the result of this methodology, this problem is solved when the 'apprentice' is a machine. Furthermore, sequential rules were chosen as a knowledge representation just so that the resulting microstrategy would be easy to communicate to people.

"Expertise may not be applicable even when expressed in language. An apprentice may not be able to convert verbal comprehension of the basis of a skill into skilled performance." In the statistics domain, the models produced are completely operational. Given the prescribed inputs for the decision, a program describes how to reach a decision. In other domains, a point of required human activity might be producing the variables used in the rules. Furthermore, non-symbolic tasks, such as riding a bicycle, are simply outside the domain of applicability of this methodology, as they are for most knowledge acquisition methodologies used for knowledge based systems.

"Expertise expressed may be irrelevant. Much of what is learnt, particularly under random reinforcement schedules, is superstitious behavior that neither contributes not detracts from performance." Given the seven models that I have built with with methodology, it is possible to study this question empirically. We can systematically remove each rule in turn from each model and see if it continues to do as well. In fact, no rule is irrelevant by this test. The range of degradations of performance is from 4% of the (training) examples to 40%. Apparently the method avoids random reinforcement schedules.

"Expertise expressed may be incomplete. There will usually be implicit situational dependencies that make explicit expertise inadequate for performance." This point is related to the first one, and both the grey region and example selection mentioned there are relevant here also. However, it is certainly true, even on the training examples, and even with judgements changed for better consistency, that the models do not exactly reproduce the judgements. Surely, then, as more examples are encountered, there will be some errors, and there may indeed be rules needed to cover cases not represented in the examples studied. However, if some representative way of gathering examples is followed, and if about 100 examples are gathered, then the cases in which additional rules are needed should be rare.

"Expertise expressed may be incorrect. Experts may make explicit statements which do not correspond to their actual behavior and lead to incorrect performance." Since language is not used in a performance based methodology, then this problem is solved. The knowledge representation is not uttered directly, but is inferred from examples of the required expertise.

3.2 Expressive Adequacy of the Knowledge Representation

The previous section reduced the question of the expressibility of expertise to the question of the expressive adequacy of the knowledge representation. A related question is whether the interactive process, which allows changing judgements and adding new statistics, is abused to accommodate an insufficiently expressive knowledge representation.

Having built several different microstrategies, I can begin to address these questions empirically. The following table shows some attributes of a group of seven decisions in graphical presentation and of the models of the decisions.

Table 1
Attributes of Decisions and Models

model identifier	inputs	alternatives	statistics	1 hyp rules	2 hyp rules	rule targets	target sequences
ts-scale	2	3	3	1	1	3	3
ts-super	2	2	4	4	0	1	1
xy-scale	2	3	5	3	1	2	2
ts-zero	1	2	1	1	0	1	1
ts-smooth	1	2	4	4	0	1	1
ts-lines|sm	1	4	3	2	1	2	2
ts-lines|nosm	1	4	3	2	1	3	3

The models named in the table all address decisions of graphical presentation. The "ts-scale" decision is whether to use the same scale, the same units per centimeter, or different scales when plotting two equal length time series in juxtaposed panels. The "ts-super" decision is the one whose model is presented above, whether to superpose two time series in one plot or to juxtapose them in two plots given that they will be plotted against the same scale. The "xy-scale" decision is whether to use the same scale, the same units per centimeter, or different scales when plotting two variables in an x-y scatter plot. The "ts-zero" decision is whether to include zero in the vertical scale when plotting a single time series. The "ts-smooth" decision is whether to include a specific smooth with a plot of a single time series or not. The "ts-lines|sm" decision is whether to use a symbol plot, a connected symbol plot, a line plot, or a vertical line plot, given that the smooth will be included with the plot. The "ts-lines|nosm" decision is whether to use a symbol plot, a connected symbol plot, a line plot, or a vertical line plot, given that the smooth will *not* be included.

The order of the models in the table is the order in which they were built. The first two were built by hand, the others with automated assistance. The inputs to the decisions were either one or two vectors of real numbers. The number of alternatives ranged from two to four. The number of statistics cited in the rules is the dimensionality of the model. This ranged from one to five, with a mean of about 3. 80 percent of the rules have one hypothesis, and only 20 percent have two. Since one hypothesis rules were much more common than two hypothesis rules, we might extrapolate that the need for three or more hypotheses would be even less. This gives some indication that no more than two hypotheses are needed.

With one exception, the number of rule targets is less than the number of alternatives. This means I was able to build models which specified reasons for using each alternative besides the default. The number of rule target sequences is equal to the number of rule targets in each case. This means that I was able to give rules that would pick out all the examples for one alternative before going on to others. Both of these restrictions make the model easier to understand. I considered such models desirable, and in some cases built more than one model in order to achieve this end. That I was able to build such models, imposing a further restriction on the form, also suggests that the formalism is sufficiently expressive, at least if used interactively.

With the freedom to change judgements, and to define new statistics, a question that has concerned me has been whether the models are overfitting the judgement data. I collected some attributes of the model building process to give some insight into this question. The following table shows how many statistics were used at various stages of the modeling process.

Table 2
Use of Statistics

model identifier	statistics used	statistics defined	initial statistics	statistics added
ts-scale	3	7	7	0
ts-super	4	7	3	4
xy-scale	5	6	6	0
ts-zero	1	3	3	0
ts-smooth	4	6	5	1
ts-lines|sm	3	4	3	1
ts-lines|nosm	3	3	2	1
totals	23	36	29	7

The data columns show the number of statistics used in the final model, the total number of statistics defined while building the model, the number of statistics defined after initial judgements but before looking at any judgement plots, and the number of statistics defined after looking at one or more judgement plots. Most (65 percent) of the statistics ever defined were used in the final models. Few (20 percent) of the statistics were defined after looking at the judgement data. These observations suggest that the freedom to define new functions is not being abused to get models fitting into the knowledge representation.

The following table shows how many judgements were changed.

Table 3
Changes in Judgements

model identifier	examples used	final preference			final alternative		
		same	upgrade	new	downgrade	same	new
ts-scale	100	99	0	1	1	0	0
ts-super	81	81	0	0	0	0	0
xy-scale	100	96	0	4	2	11	6
ts-zero	92	92	0	0	0	11	1
ts-smooth	123	120	1	2	1	20	7
ts-lines|sm	72	66	0	6	1	28	7
ts-lines|nosm	51	49	2	0	2	17	2

The second column shows the total number of examples used to develop the model. Each example has a preferred judgement, and some examples have acceptable secondary alternatives. These judgements may vary during the model building. The third, fourth and fifth columns refer to the final preferred judgements for each of the examples. The third column shows how many of the final preferred judgements were the same as the initial judgements; the fourth column shows how many final preferences were initially acceptable secondary alternatives; the sixth column shows how many final preferences had not been included initially. The sum of these three columns equals the number of examples. The final three columns refer to the final acceptable secondary alternatives, when they exist. The "downgrade" column shows how many were initially the preferred alternative. The "same" column shows how many were also initially considered an acceptable secondary alternative. The "new" column shows how many were not initially either preferred or an acceptable secondary alternative.

One difference introduced by using software assistance which supported alternative judgements was that I started using them. I kept only one acceptable alternative out of the 181 judgements in the two models built by hand (the first two listed), but gave alternatives for about 30 percent

of the examples used in the models built with machine support (the last five). Excessive use of alternatives would make the models based on them less meaningful. However, to the extent that judgements are not definitive, then a model may well be stronger for having the extra information.

I think that changes in the judgements are potentially more dangerous than excessive use of alternatives. The table shows that the preferred judgement was changed in only three percent of the examples (sum of "upgrade" and "new" compared to total examples). A new secondary alternative was introduced for five percent of the examples. While I do not have a standard for evaluating these levels of potential data fudging, they do not seem high.

Both judgement changing and statistic definition are an intrinsic part of the interactive knowledge elicitation process. Thus, we can also look at the data in Tables 2 and 3 as indicating the extent to which there is knowledge to elicit beyond just collecting initial judgements and running totally automatic modeling software. Viewed in this light, the judgement changes and additional statistics defined suggest that automatic model building would miss some information.

Another way to examine the expressive adequacy of the representation is to compare it to other possible representations. I considered two alternative kinds of models, a tree model and a logit model, which I will describe shortly. To distinguish the method described in this paper from the automated methods, I will call it the sequential model. I compared all three models on their ability to fit the final preferred judgements, since this is the kind of data that is commonly used for automatically fitted models. By construction, the sequential models provided a perfect fit to preferred or secondary judgements using the loose limit. However, it is not necessarily a perfect fit or even the best fit to the preferred alternative. For automated use, the sequential model has a free parameter, the fraction between the loose and tight limits to use as the decision point. I selected the fraction which minimized the errors on the final preferred judgements. The comparison shown here is thus best seen as a comparison of the ability of the representations to fit the final judgements.

One of the alternative models was a tree model. I used a tree modeling package available in S (Pregibon, 1988). The algorithm makes a standard binary recursive partition of the data, seeking a maximal differentiation of the two partitions at each step. The result is a series of trees with an increasing number of rules achieving an increasingly good fit to the training data. I selected the tree model with the same number of rules as the sequential model had hypotheses. This equates the number of hyperplanes used to divide the data in the two methods.

The other alternative model was a less standard logit model. For a model of a decision with N alternatives, I built a series of N-1 logit regressions. Each regression used all the statistics defined in building the sequential model as explanatory variables. Each regression fit a binary dependent variable of the form alternative I versus any other alternative. The sequence of alternatives selected was decided automatically by trying each possible alternative and selecting the one which maximized the number of examples on which the alternative was (correctly) selected less those on which the alternative was (incorrectly) not selected. The alternative not selected by any of the rules was taken as the defalut decision if none of the N-1 rules were satisfied. This model allows the freedom of decision hyperplanes which are not orthogonal to an axis, but only allows one rule for each alternative other than a default.

The number of errors for each model is shown in the following table. The minimum number of errors for each model is followed by an asterisk.

Table 3
Comparing Representations:
Errors in Fitting Final Preferred Judgements

model identifier	sequential model	tree model	logit model
ts-scale	0*	5	4
ts-super	0*	0*	4
xy-scale	7	9	6*
ts-zero	2*	2*	3
ts-smooth	18*	21	24
ts-lines|sm	12	10*	13
ts-lines|nosm	6*	8	6*
totals	45	55	60

The table shows that while the sequential model led, it did not dominate either of the other methods. It had the least examples misclassified in five of the seven cases, while the tree models achieved the minimum in three of seven cases and the logit model achieved the minimum in two of seven cases. The total errors was the least for the sequential model, followed by the tree model and logit model. Each of the tree model and logit models achieved a better result than the sequential model in one of the seven cases.

That the sequential model leads in this comparison suggests again that the representation is not unduly restrictive. That it does not dominate the other two suggests that making their results available also could be a useful addition to the software tools available.

3.3 Summary: Support of Knowledge Acquisition

To summarize this discussion, the methodology described here supports knowledge acquisition is several ways. A simple way it is supported is through accepting notes as preliminary judgements are made. It is readily apparent when an example shows some novel feature and when it shows only features that have been though about before. When a novel feature is presented, it is easy to make a short note about it. Collecting these notes afterwards, and sorting them by the preferred alternative provides a useful set of ideas about some statistics to define. The process of making a set of judgements thus becomes the first stage of making the basis for the judgements formal.

The method places primary reliance on plots of judgements against potentially interesting statistics. Visual inspection of the plots shows whether the potentially explanatory statistics can be used as the basis for a rule. This primary reliance is shared by various non-interactive rule induction techniques. Two specifics of the technique highlight key examples to review carefully. First are examples which are similar on the basis of the statistics currently being examined, but which have differing judgements. Second are examples which form the marginal cases for placement of rule boundaries.

*Re*examining these key examples is the key to the additional knowledge acquired by having an interactive technique. The methodology and software allow several responses in reexaming an example. First, and simplest, the judgement may simply be changed. This amounts to confirming that the currently considered statistics are satisfactory representations of the basis for making the decision. Second, the judgement may be reconfirmed. This is the strongest case. It amounts to a clear statement that the currently considered statistics are not a satisfactory basis. In this case, it is appropriate to define an additional statistic. A third response may be to accept a new secondary alternative. This is the weakest response. It

amounts to a weakening of the force of the currently considered statistics, which will be reflected in a wider grey region for the rules.

4. Uses of Microstrategies

Microstrategies may be widely useful. Here I sketch three broad classes of use. These are the use of microstrategies as a vehicle for studying knowledge acquisition, as a means of generating useful formal models, and as a means for improving tutorial material.

4.1 The Study of Knowledge Acquisition

Since most knowledge-based systems are large in scope they require considerable commitment on the part of the expert modeled. It is essentially impossible to replicate the study. A microstrategy can be built in a few hours, so there is some hope of getting the cooperation of enough experts to replicate studies. Microstrategies could then form the methodological basis for empirical studies of various issues in knowledge acquisition. This use of microstrategies is, of course, totally divorced from the specific methodology that I have presented here.

A simple example is shown by the study in Section 3.2, comparing the sequential model and manual fitting with a tree model and a logit model with automatic fitting. That is, two different knowledge acquisition methods can be directly compared on how they perform for deriving models of simple decisions. A point that could use more thought is how the two methods should be compared. To the extent that differing methods provide complementary information, then simple descriptions of the information each provides would be useful comparisons. However, the comparison of competitive methods is more difficult. By definition, there is no agreed standard for the decision being considered, so models derived by competitive methods cannot be compared to an agreed model initially. Perhaps the study could plan to develop more refined models of the given decision. Then the results of several initial formalization methods could be compared to the final model.

How to combine models of the same task from different experts needs to be studied. There is no current experience with this. Yet the generality of a model of just one person is always open to question. Models of simple decisions offer a way to study how to combine decision models of individuals into a single model acceptable to all. One approach to combination is mechanical. One might use factor analysis to establish dimensions represented by the combined statistics of all the models. Rules on these dimensions could then be fit to weighted judgements on each of the examples. Social processes such as the Delphi technique [Dalkey, 1969] and Nominal Groups Technique [Delbec, 1971] have been used to estimate scalar quantities, such as failure probabilities. There is no experience with combining structured objects, such as even the simple models of the microstrategies. However, analogues to the social processes for scalar estimation could be devised, giving each participating expert feedback on the other models, and then allowing the experts to revise their models. The absence of convergence would be met in the short run by a mechanical combination, and in the long run by research on the questions sharpened by the comparisons.

Some conclusions of a survey [Stillwell, 1982] of expert judgement (of scalar quantities) are interesting. First, "In a large number of studies, small or no differences have been found in the quality of the group product among the aggregation procedures compared." This generalization includes both mechanical and social techniques for aggregating decisions. Since mechanical techniques are simpler and cheaper, the observation would favor them. This is probably sufficient reason to develop them first. However, another conclusion is that people do usually put higher credibility on the results of social combinations. I think mechanical techniques would be best used as part of some deliberate social process, as a method of feedback to the

participants. For when the *structure* of a model is in question, one might expect mechanical procedures to miss an important idea suggested by only one person, but which others could readily agree to once it had been pointed out. And indeed, given interactive construction, it might well happen that a final model embodied ideas not present in any initial model.

4.2 Useful Formal Models

By reviewing the current applications of this methodology, other possible applications may be suggested. The statistics domain has some special properties that I am used to exploiting. Especially, the ability to simply define a new statistic and get values for all the examples present automatically. However, with suitable modifications, I believe this methodology and some or all of the software could be useful in other domains.

The heaviest use of the methodology has been to define preliminary models in questions of graphical presentation (such as the model shown in Section 1). The intended use of these models is as defaults for plotting routines. With the ubiquitous use of computers, there are questions of how to present results to users in every occupation in the society. The answers to these questions are not formalized now, and doing so would probably be of great value. Their provision as a default means that one need not entertain such heavy burdens of responsibility (legal and ethical liability, for instance) as to delay getting the best current models into practical use. In the majority of cases the computer's output is numerical. The software described here would be precisely applicable. Judgements collected from a sample of prospective users could provide an empirical basis for a number of decisions that are now made subjectively and perhaps politically by a programmer or a group.

A nearby set of domains preserves the ability have new statistics calculated automatically, although the process may involve more data than I have used. For example, speech and text recognition both use digitally recorded inputs. A second use illustrates this kind of application. A text recognition program under development at Bell Labs uses automatic learning on high level features derived from a bitmap of the character. The learning is across examples confounded by synthetic noise representing common printing errors, such as ligatures, spaces, and thickened or thinned lines. While it achieves a high level of accuracy, it still has some confusions. A study of the recognition of "J" (capital jay) and "]" (right square bracket) was made to assist in development of the system. The first question was whether a person could distinguish the letters the machine was confusing. A quick inspection of the bitmaps showed that it was always possible, and that the upper right serif on the capital jay was diagnostic.

The second question was whether a person could distinguish the characters on the basis of the abstraction available to the machine. This consisted of vectors tracing the outline of the character and of vectors showing concavities in the traced outline. It was easy to transfer a sample of fifty examples of each character to S, and to have the examples presented in random order. The problem for the machine soon became apparent, because the upper right serif was not traced with a concavity in all cases. However, it was possible to distinguish all the examples perfectly, because when the serif was not represented by a concavity, then the two long lines were not parallel. A model based on the number of concavities and the angle between the two longest lines provided a perfect fit to the judgements. The model was not of direct use in this case, but was the basis for recommendations to the developer. The first recommendation was that the boundary tracing routine should be examined to see if the serif visible by eye could be caught by a concavity always rather than most of the time. The second recommendation was that the angle between the two longest lines was a high level feature that might be useful to the automatic learning mechanism. Further studies are under way.

Other examples of areas that seek to interpret numerical inputs are voice and speech recognition, and instrument readings in medicine or science. The use of interactive techniques is probably quite appropriate in medical interpretations, where the ground truth is not so certain. While electrocardiogram interpretation has been studied, there are many other instrumental readings that might profit from interpretation.

The third current use of the software is for interpreting a particular plot in statistics. The intent is to generate automatically a legend to accompany the plot. The legend is to consist of several sentences each of which will report on one decision. Analogous legends might be a useful means to present computational results in well understood situations.

A further set of domains loses the ability to define new statistics easily, but can still use the methodology. An example might be the diseases of a plant species which could be diagnosed from leaves of the plant. A collection of photographs could be built, and initial attributes defined. If the judgement plots showed a need to reexamine a judgement, then the photograph could be reexamined. If the user were convinced of the need for a new attribute, then it could be gathered, albeit with greater difficulty. The principle requirement is that the original decision materials be archivable; for if a series of on the spot decisions and attribute collections is made, then they will not be subject to review.

4.3 Application to Teaching

A common tutorial device is to state a principle and provide an illustrative example. Statement of a principle is equivalent to identifying a dimension of concern. It is more useful to also show a quantitative estimator of this dimension, and still more useful to indicate a range of critical values of the statistic. The methodology discussed here provides a natural way to go about increasing the knowledge conveyed in a tutorial discussion. It also provides a way to select key examples for tutorial presentation. The plot of the teacher's judgements for a number of examples would also have considerable tutorial value.

REFERENCES

Becker, R., J. Chambers, and A. Wilks, 1988, *The New S Language*, Wadsworth, Monterey, California.

Breiman, L., J. Friedman, R. Olshen, and C. Stone, 1984. *Classification and Regression Trees*, Wadsworth, Monterey, California.

Dalkey, N., 1969, "An Experimental Study of Group Opinion: The Delphi Method," *Futures*, v. 1, 408-426.

Delbec, A., and A. Van de Ven, 1971, "A Group Process Model for Problem Identification and Program Planning," *Journal of Applied Behavioral Science*, v. 7, pp 466-492.

Gaines, B. R., 1987, "An Overview of Knowledge Acquisition and Transfer," *International Journal of Man Machine Studies*, v. 26.

Gale, W. A., 1986, "REX Review," in *Artificial Intelligence and Statistics*, W. A. Gale ed, Addison-Wesley, Reading, Massachusetts.

Pregibon, D., 1988, personal communication. "Tree Modeling in S" manuscript in preparation.

Stillwell, W. G., D. A. Seaver, and J. P. Schwartz, 1982, "Expert Estimation of Human Error Probabilities in Nuclear Power Plant Operations: A Review of Probability Assessment and Scaling," NUREG/CR-2255.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

A Method for Acquiring Strategic Knowledge*

Thomas Gruber**

Experimental Knowledge Systems Laboratory
Computer and Information Science
University of Massachusetts
Amherst, MA 01003
gruber@sumex-aim.stanford.edu

Abstract

This paper presents an automated method for acquiring strategic knowledge from experts. Strategic knowledge is used by an agent to decide what action to perform next, where actions effect both the agent's beliefs and the state of the external world. Strategic knowledge underlies expertise in many tasks, yet it is difficult to acquire from experts and is generally treated as an implementation problem. To acquire strategic knowledge, we have designed an operational representation of strategy, devised a technique for eliciting it from experts, and built an assistant that manages a learning dialog with the expert. The assistant elicits cases of expert-justified strategic decisions and generalizes strategic knowledge with syntactic induction guided by the expert. The knowledge acquisition method derives its power and limitations from the way in which strategic knowledge is represented and applied.

1 INTRODUCTION

In this paper we present a method for acquiring strategic knowledge from experts with machine assistance, in the manner of interactive knowledge acquisition systems such as TEIRESIAS (Davis, 1976), ETS (Boose, 1986), SALT (Marcus, 1987), and MOLE (Eshelman, Ehret, McDermott, and Tan, 1987). We begin with a description of strategic knowledge, the problem of acquiring it, requirements for a solution, and a brief description of the approach. We trace an example session between an expert and the computer assistant called ASK to illustrate the functionality of the approach. We then explain how ASK works, outlining its essential components. We conclude by discussing the sources of power of the method, its scope of applicability, and its limitations.[1]

1.1 The Problem of Acquiring Strategic Knowledge

Strategic knowledge is knowledge about how to act; more precisely, it is knowledge used by an agent to decide what action to perform next, where actions effect both what is believed by the agent and the state of the external world. A robot applies strategic knowledge to decide

* This research was sponsored by DARPA-RADC contract F30602-85-C0014 and ONR University Research Initiative contract N00014-86-K-1764.

** Author's current address is Knowledge Systems Laboratory, Stanford University, 701 Welch Road, Building C, Palo Alto, CA 94304.

[1] In an earlier paper (Gruber, 1987), we reviewed the relationship of this work to the major paradigms of knowledge acquisition .

between going around or over an obstacle. A lawyer uses strategy in deciding which question to pose in cross examination and which case to cite in defense of an argument. Strategic knowledge underlies expertise in domains where managing the *process* of problem solving, choosing among possible actions, is critical to performance. It is distinguished from the *substantive knowledge* of a domain, knowledge about what to believe, such as the inferential relationships among data and hypotheses. It is heuristic knowledge for dealing with incomplete and uncertain knowledge about the world. An agent that can *act* appropriately without predicting the precise consequences of potential actions is more robust.[2]

We have studied the strategic knowledge that physicians bring to bear in the medical work-up: the process of gathering data, assessing the results, and planning treatment. In AI, diagnosis is often treated as a pure classification problem, applying substantive knowledge to identify likely causes for findings. Viewed as a process, however, the medical work-up requires additional knowledge about how to proceed when the data is not all in. Strategic knowledge underlies the physician's expertise at choosing the next question to ask, diagnostic test to run, or therapy to try.[3]

Although strategy is expertise, it is very difficult to acquire strategic knowledge from experts. The conventional separation of control from inference in knowledge systems treats strategy as an implementation problem, a responsibility of the knowledge engineer. While substantive knowledge can be acquired in declarative form, such as rules mapping causes to effects or evidence to hypotheses, strategic knowledge about choosing actions is often formulated in a way that complicates its acquisition. For example, in the MYCIN system, much of the knowledge that determined question ordering and decisions about laboratory tests was represented with screening clauses, clause ordering, and certainty factors — implementation-level manipulations of the substantive knowledge to achieve the intended strategic behavior (Clancey, 1983a). Since MYCIN, more explicit representations of strategic knowledge have been devised (e.g., S.1's control blocks (Erman, Scott, and London, 1984)), yet the implementation of strategy in knowledge systems is still largely performed by knowledge engineers.

For domains where strategy plays an important role, it is less productive to "engineer" strategic knowledge than to acquire it directly from experts. First, if strategic knowledge is represented with implementation-level primitives, such as numeric priorities, the order of rules, or sequential programs, then the usual benefits of explicit knowledge representation are lost: explainability, modifiability, and meta-level reasoning. Second, excluding the expert from participating directly in the formulation and debugging of strategic knowledge is inefficient for expert and engineer alike. We believe the difficulty of acquiring strategic knowledge is a limiting factor in the dissemination of knowledge system technology today.

2We have defined strategic knowledge more narrowly than the general term "control knowledge," emphasizing that strategic choices concern external actions. *Search control* knowledge directs the internal events in problem solving, such as the choice of an operator or the firing of a rule. The distinction hinges on the purpose to which the control knowledge is put. In tasks with a complete predictive model of the consequences, a good evaluation function, and no resource limitations, exhaustive search can be used to project into the future and decide how to act accordingly. Search control knowledge can improve an evaluation function and speed up search.

3This project has been motivated and informed by experience in building a knowledge system for the prospective diagnosis of chest pain, called MUM (Cohen, Day, Delisio, Greenberg, Kjeldsen, Suthers, and Berman, 1987). MUM's performance criteria were not only to get the right diagnosis, but to choose diagnostic actions as a physician would, asking questions in an intelligent order and balancing the potential costs of diagnostic test and trial therapy with the evidential and therapeutic benefits.

A primary goal of the research reported here is to devise a method for acquiring strategic knowledge from experts that can be *automated*, which would facilitate the construction of knowledge systems that act with good reason. We present the method and then discuss the design of the implementation, showing how general principles were applied to this knowledge acquisition problem.

1.2 Objectives and Overview of Approach

What constitutes a solution to the problem of acquiring strategic knowledge? Initially, we must characterize the behavior that we wish to model and identify the knowledge that is sufficient to produce the desired behavior. We define strategic knowledge as that used iteratively to decide which actions are appropriate in a given situation. This is a theoretical commitment to a particular style of strategic reasoning that has been called reactive planning (Chapman and Agre, 1987; Firby, 1987). Other ways of viewing strategy are possible, such as skeletal planning or backtracking search. We choose the reactive view of strategy and a particular representation to facilitate knowledge acquisition; the rationale for and the implications of this design decision are discussed in Sections 4 and 5.

Given a general view of strategic knowledge, we must design or adapt an operational representation that can be executed by a knowledge system to achieve the desired behavior. In practical terms this means designing an architecture for control. We have chosen to represent strategic knowledge with *strategy rules*, inspired by the metarules that represent diagnostic strategy in NEOMYCIN (Clancey, 1984a; Clancey and Bock, 1988). A knowledge base of strategy rules serves as the control component of a knowledge system. Strategy rules map strategic situations to sets of actions with specified features. At each iteration in a running system, strategy rules recommend the actions to perform next. The strategy rule representation is described in Section 3.1.

In addition to an operational representation, we need a technique for eliciting the strategic knowledge from people. A knowledge elicitation technique has two main functions: to ask for the knowledge and to provide a means for the expert to answer. In theory, one could ask the expert to "tell me your strategy" and provide the operational representation as the language in which to reply. In practice, an elicitation technique must constrain the dialog, so that the system only asks for knowledge it needs, and asks questions in a form that is natural to the expert. Our experience indicates that experts can give *justifications* for their actions (Cohen et al., 1987). This influences the design of the elicitation medium: it should be a language for describing the rationale for specific strategic decisions. Our system asks the expert why an action was chosen in the context of an actual case, and the expert answers by indicating which factors were relevant to the current choice. The system provides a set of factors to choose from, and if the currently known factors are incomplete, the expert is given an interface for defining new ones, also in the context of justifying actions. The elicitation technique is described in Section 3.2.

Finally, if we are to *automate* the acquisition of strategic knowledge we need to build an assistant that can structure the dialog with the expert and formulate strategy in the operational representation. The assistant needs to know what knowledge is required, how to elicit it from the expert, and how to put it in operational form (i.e., in the operational representation). We have developed an assistant called ASK that elicits justifications from the expert and generates strategy rules. It uses simple inductive operators to generalize what it has acquired, and asks the expert to verify the generalizations.

Details about the representation, the elicitation technique, and the learning process are presented in Section 3. First is a demonstration of the method: a sample knowledge acquisition dialog.

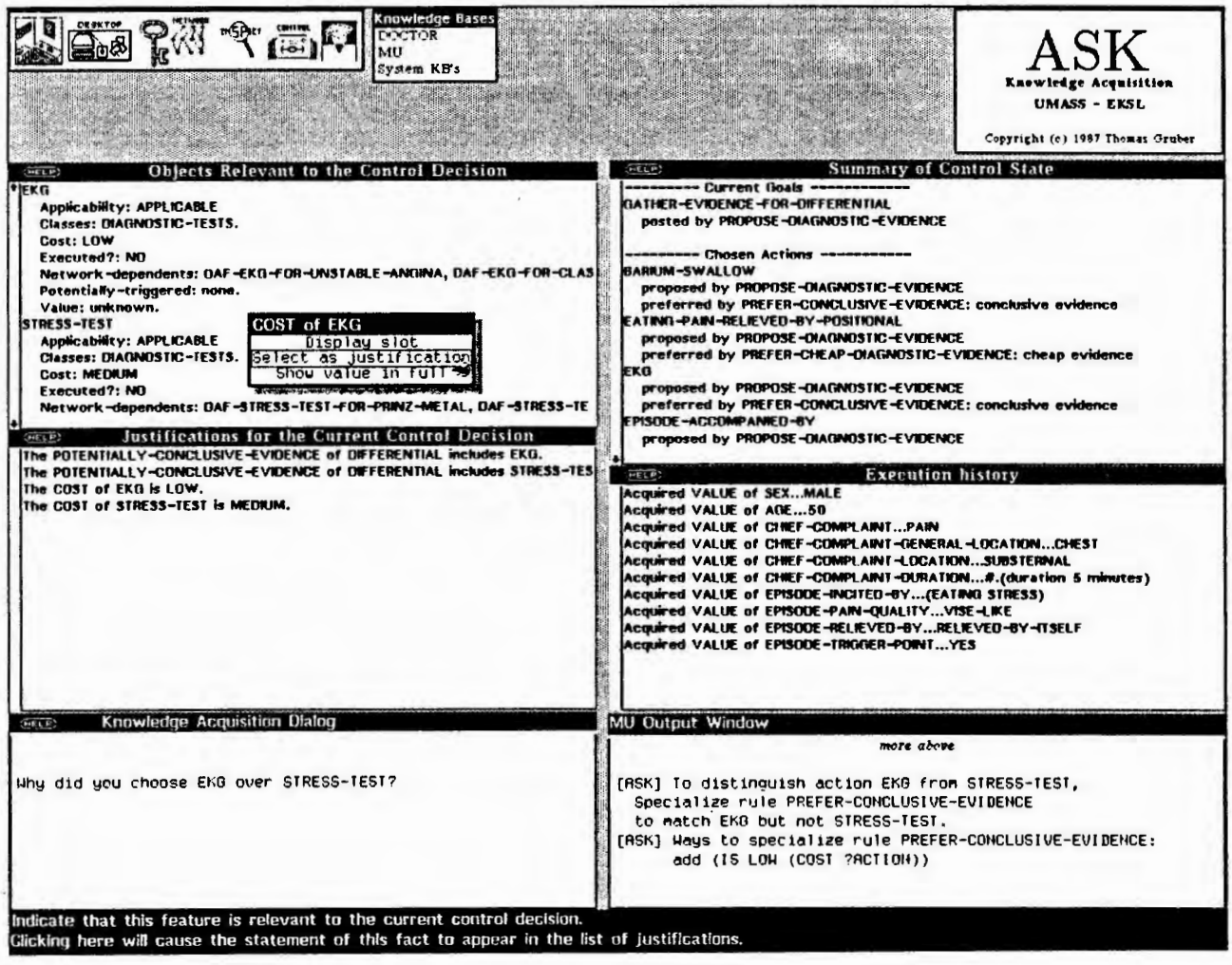


Figure 1. A snapshot of ASK's justification interface.

The justification interface is a means for the user to justify or explain why a particular strategic decision was made. Text in the windows is mouse sensitive; by clicking on words or lines, the expert can access to menus of operations on the entities represented by the text. The expert justifies a choice of an action by selecting features of relevant objects, which adds sentences to the contents of the justifications window. In this case, the expert has clicked on the fact "Cost: LOW." under the object EKG in the upper left window; the sentence "The COST of EKG is LOW." appears in the justification window below. The justification interface shown in this screen is integrated with a general knowledge base browsing facility; windows and graphic displays for that facility are not shown here. The windows on the right of the screen show the state of the application and internal trace information. The icons at the top give access to menus of commands for running the application and developing the substantive knowledge base.

2 AN EXAMPLE DIALOG SESSION WITH ASK

This section presents an annotated transcript of a knowledge acquisition session between the expert and ASK. Annotations in *italics* explain what the expert and program are doing. For economy and clarity it is written as a natural language dialog, with ASK prompting for information. In actuality, the expert answers by clicking on items in windows and menus, and the user can take the initiative in the dialog. A snapshot of the screen is shown in Figure 1.

2.1 The Setting: A Work-up for Chest Pain

The dialog begins during the execution of a knowledge system for the diagnosis and treatment of chest pain. The system already has the substantive knowledge of the domain relating the potential effects of data-gathering actions (history and exam questions, diagnostic tests, and trial therapy) to hypotheses (diseases and related conditions).[4] The system has been supplied with a small amount of strategic knowledge, sufficient to tell it how to get the initial data on a patient (by asking questions) and how to gather evidence for hypotheses that have been triggered.[5] It can therefore generate a set of possible actions that it might take to investigate the suspected diseases of the current patient. But it doesn't know *which* action to take next.

The expert has selected a case in which the patient has symptoms that suggest various kinds of angina, and the knowledge system has run to the point where an interesting strategic decision must be made. The expert wants to teach the system how to choose its next action in this situation.

2.2 Eliciting Justifications for a Strategic Choice

ASK: `In this situation, I would choose one of:`
`EKG, Stress-test, Vasodilator-Tx, Angiogram.`

The existing strategic knowledge causes the chest pain system to propose these four actions, but it cannot decide among them.

User: `I would choose EKG.`

ASK: `Why? They are all equivalent to me.`
`(I suggested them because they were proposed for the goal called gather-evidence-for-differential, and because they are in the potential-evidence of the differential.)`
`Please provide justifications that distinguish EKG from one of the others.`

ASK analyzes the current state of the strategic knowledge and determines that in order to account for the expert's choice, EKG, it should acquire justifications that distinguish EKG from the other actions. ASK explains why the system chose the four actions, and asks the expert for his or her justifications for selecting EKG over others.

User: `I would prefer EKG over Stress-test because`
`"EKG potentially-confirms classic-angina."`

---

[4]The substantive knowledge was previously acquired for the MUM system (Cohen et al., 1987).

[5]This session started with 14 strategy rules that represented a general diagnostic strategy. ASK can be used without any prior strategic knowledge, but this example demonstrates the acquisition of domain-specific knowledge to refine a task-specific strategy.

The expert justifies the choice of EKG by selecting from facts about the current situation and features of EKG. The "potentially-confirms" relation between EKG and one of the active hypotheses, classic-angina, is a fact computed dynamically by the inference net. The set of possible facts are provided by ASK in a browsing interface, and the expert searches through the possible facts to choose those that are "relevant to the current strategic decision." The selected facts serve as justifications for choosing EKG in this situation.

ASK: "potentially-confirms" does not distinguish EKG from stress-test. Please provide further justifications for choosing EKG.

The potentially-confirms relation also happens to hold for the other three actions; they all potentially provide conclusive evidence for hypotheses on the differential.

User: I would prefer EKG because "The cost of EKG is low."

With this justification, ASK can form a rule that distinguishes EKG from Stress-test. The rule says, roughly, "In this situation, select actions that are potentially-confirming and have a low cost." This rule reduces the set of chosen actions to two.

ASK: If I consider "potentially-confirms" and "cost", I would choose EKG and Vasodilator-Tx. Is this an improvement?

User: Yes.

ASK: Would you still choose EKG over Vasodilator-Tx? If so, please justify.

The expert may be satisfied with the new set of actions. If the expert has reasons for being even more selective (to teach the system to select only EKG in this situation) then more justifications need to be given.

User: Yes, because of the time-required.

2.3 Acquiring a New Feature

ASK: Please define "Time-Required". I'll Help you...

ASK does not know about the term "time-required," and so it helps the expert define it as a new feature.

ASK: What kind of feature is time-required? Is it an attribute of actions, (like "cost"), a class of actions (like "diagnostic-tests"), a new object (like "differential"), an inferential value computed by rules (like "level-of-support"), or a dynamic relation (like "potentially-confirms").

User: attribute.

ASK plays the role of knowledge engineer, mapping the expert's construct to the representational primitives used in the knowledge base. This question about the type of the new feature is vague when couched in implementation-level terms such as "dynamic relation," so ASK provides instances from the current knowledge base as examples, such as "potentially-confirms."

ASK: EKG is a member of the classes Diagnostic-tests, Data, and Actions. To which of these classes of objects will the attribute Time-required apply?

User: Diagnostic-tests.

The domain of a feature is a class. Since the expert is defining this term to describe the object EKG, ASK knows that the domain of the new feature must be some parent class of EKG, and it prompts accordingly (with the graph of a small fragment of the knowledge base hierarchy). The context in which this feature is acquired allows ASK to constrain the interface much more than a general-purpose knowledge engineering tool.

ASK: What possible values might time-required take?
(e.g., The possible values of Cost are "free, cheap, low, ...")

User: immediate, few-minutes, an-hour, few-hours, a-day, few-days, ...

ASK: Is there an ordering over the values, such as "Immediate ≤ few-minutes ≤ an-hour ..."

...

After the new feature is well-defined, the ASK provides an interface for specifying the value of Time-required on the members of the class Diagnostic-tests (or letting them inherit a default value).

When the expert is finished, the knowledge base has a new feature that can serve as a justification for selecting actions — in this case, the Time-required distinguishes EKG from Vasodilator-Tx.

2.4 Formulating and verifying a new strategy rule

ASK: Considering "potentially-confirms", "cost", and "time-required", I would choose EKG, too. I would paraphrase your advice as:

Select an action if:
a goal is to gather-evidence-for-differential, and
a hypothesis is in the differential, and
the action potentially-confirms the hypothesis, and
the cost of the action ≤ low, and
the time-required of the action = few-minutes.

Is this what you mean?

User: Yes.

ASK has formulated a new strategy rule based on the expert's justifications for choosing EKG. It is based on the existing rule that proposed the original four actions. It maps the situation identified by the goal and the state of the differential to an equivalence class of acceptable actions — those that match the last three clauses. This rule causes the system to choose low cost, quick, potentially confirming actions when the goal is to gather evidence for hypotheses on the differential. Note that the rule captures a tradeoff : it says that actions that cost "low" (which is higher than "free" and "cheap"), are acceptable if they are potentially confirming and only take a few-minutes to run. The Vasodilator-Tx action can take weeks.

2.5 Generalizing the New Rule

The new rule is operational but specific to actions that are very similar to the training example (i.e., to actions that take exactly "few-minutes"). ASK asks hypothetical questions to try to generalize the rule to cover actions that are near misses to EKG. Since it doesn't have any common sense, ASK has to ask whether it is better if tests take less time.

ASK: What if the time-required of EKG were immediate?
Would you still choose it?

User: Yes.

ASK: What if the time-required of EKG were an-hour?

User: Yes.

ASK: What about few-days?

User: No.

ASK tries to generalize the rule by making it match more actions. In this example, ASK checks for thresholds that would bound acceptable values of the Time-required feature. The rule is generalized to apply to actions that require times from "immediate" to "an-hour."

After verifying the generalized rule, ASK continues the execution of the knowledge system. On the next iteration, the system would choose EKG, and if it failed to produce conclusive evidence, it would then suggest Vasodilator-tx, and so on.

2.6 Summary of the Dialog Session

In this dialog, ASK acquired strategic knowledge about how to choose among diagnostic tests and therapies. It elicited justifications for action choices from the expert, in the context of a set of possible next actions in an actual case. To extend the justifications, the expert defined a new feature, with ASK's assistance. ASK used the justifications to build a strategy rule that refines an existing rule. It generalized the new rule with the expert's guidance. In similar strategic situations in the future, the system will be more selective in choosing diagnostic actions, picking them in an order reflecting the expert's evaluation.

3 HOW IT WORKS: THE DESIGN OF ASK

3.1 Representing Strategic Knowledge with Strategy Rules and Features

Recall from our definition that strategic knowledge is used to choose the next action to perform. The model of computation underlying this view is iterative control, where a system decides what to do next at each iteration of a control cycle. The MU architecture, upon which ASK is built, is designed to support this model (Cohen, Greenberg, and Delisio, 1987). MU distinguishes the substantive knowledge of a domain, concerned with assessment, from the control knowledge, concerned with action. This division of knowledge is typical of architectures for control (cf. Hayes-Roth, 1985). MU organizes the substantive knowledge as a symbolic inference network, where inference is propagated from evidence to hypotheses by local combination functions. MU represents the control knowledge as strategy rules.

Figure 2 illustrates the operation of MU's control cycle. At each iteration, strategy rules recommend a set of actions, and one action is picked arbitrarily and executed. Its effects are then propagated through the inference network, changing the state of working memory. In the chest pain system described in the example session, an action such as EKG typically produces some data that can change the evidential support in hypotheses such as classic-angina. The job of the strategy rules is to recommend a set of actions that is appropriate to the strategic situation, as represented by the current state of the inference net.

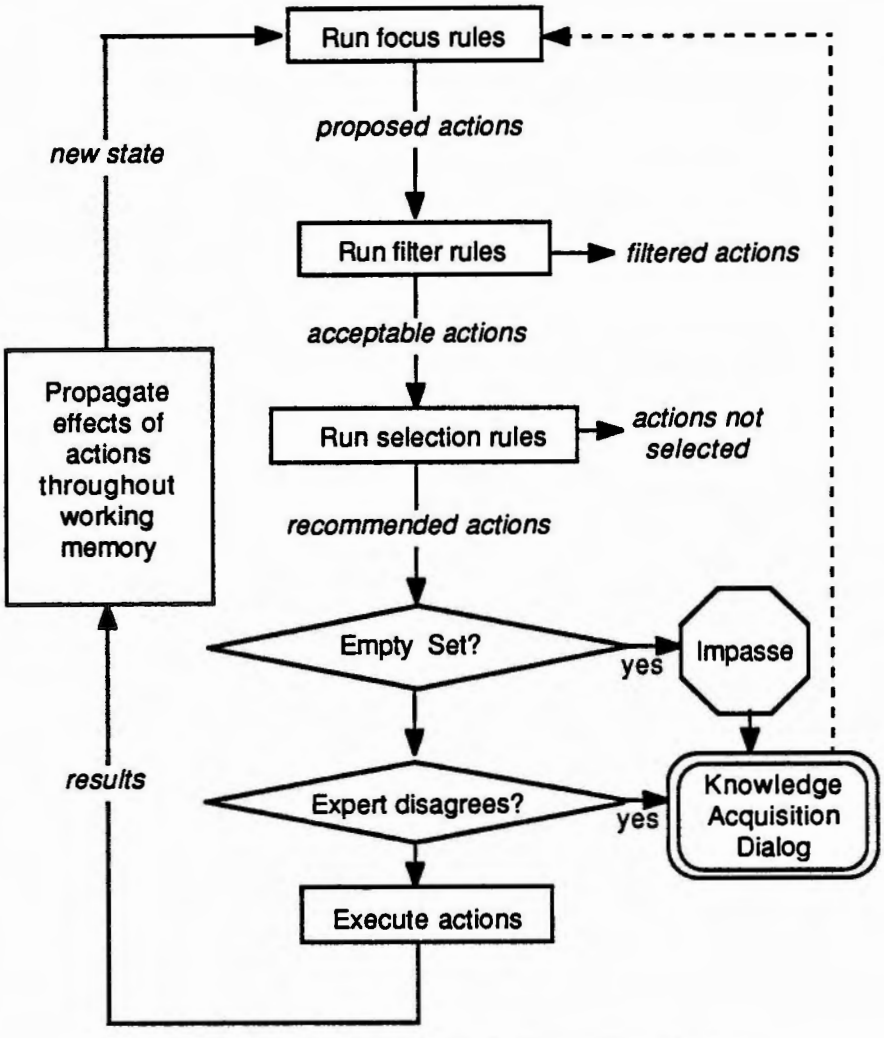


Figure 2. The Strategy Rule Execution Cycle

Strategy rules associate strategic situations with sets of actions to perform. Strategic situations are represented as expressions over attributes, functions, and relations of objects in working memory, collectively called *control features* because they specify the conditions upon which control decisions are made. The left hand sides of strategy rules are conjunctions of tests on control features; the right hand sides make recommendations about actions. Three kinds of recommendations are made. *Focus rules* examine the general state of problem solving, such as the list of active hypothesis (the differential), then post goals and propose possible actions to achieve them. In the chest pain domain, one focus rule says "If the differential is empty, post the goal generate-hypotheses and propose actions that can potentially trigger hypotheses." (The set of hypotheses potentially triggered by a data-gathering action is computed by the inference net). *Filter rules* represent general constraints on the applicability of actions. A simple filter rule from the medical domain is "Filter

actions that are contraindicated for the current patient."[6] *Selection rules*[7] represent the characteristics of actions that make them appropriate for a given strategic situation. Each selection rule specifies an equivalence class of acceptable actions. A selection rule learned in the example dialog is

```
IF (in gather-evidence-for-differential (current-goals))
   (in ?hypothesis (differential))
   (in ?action (potentially-confirmed-by ?hypothesis))
   (<= (cost ?action) low)
   (<= (time-required ?action) few-minutes)
THEN (select ?action)
```

This rule says, when trying to get evidence for hypotheses on the differential, select actions that could potentially confirm hypotheses and that do not cost much or take much time. (The terms `current-goals`, `differential`, `potentially-confirmed-by`, `cost`, and `time-required` are control features.)

At each iteration of the control cycle, strategy rules are run in a three step sequence: first, focus rules produce a set of possible actions; next, filter rules eliminate those that violate constraints; and then selection rules choose subsets of the remaining actions that are appropriate for the current situation. The union of selected subsets are recommended for execution.[8] There is no voting or weighting scheme; an action selected by one rule is equivalent to one selected by many rules. To resolve matching conflicts, rules are applied in an order specified by a precedence relation called *shadows*, which is roughly equivalent to a partial order based on generality.[9] If a rule succeeds, then the more general rules that it shadows are pruned. The shadows relation is used to represent relative selection criteria in the absence of a continuous utility function. A family of selection rules, in which the more specific rules shadow the more general, has the effect of choosing those actions judged to be acceptable by the most specific (most constraining) rules.

If the set of recommended actions is empty, then the system enters a knowledge acquisition dialog, since it has insufficient strategic knowledge to continue.[10] If the chosen set of actions is a singleton, then the system executes that action. Otherwise an action is picked arbitrarily (e.g., by the user or randomly), and executed. Thus the performance system is always potentially underdetermined, so that it can still run with *partial* strategic knowledge. With no strategy rules at all, the control of the system is left entirely to the user, who picks from the list of all possible actions.

---

[6]Whether a particular action is contraindicated is computed by rules in the inference network relating patient characteristics with therapeutic actions.

[7]In previous reports, they were called "preference rules," but the term seems to have confusing connotations.

[8]If *no* selection rules apply, then all of the proposed rules are recommended. This allows the system to run with an incomplete knowledge base of selection rules.

[9]Rule A shadows rule B if B uses a proper subset of the control features that A uses or if B is less restrictive on the same control features. For example, a selection rule testing the condition "time-required = few-minutes" shadows the same rule with the condition "time-required ≤ an-hour." Precedence between two rules that are not ordered by generality can be specified explicitly (i.e., as a relation in the knowledge base); this is necessary to resolve conflicts between tradeoffs. For instance, if it is better to choose actions with low cost and high risk over actions with moderate cost and moderately high risk, then the selection rule representing the former tradeoff should shadow the rule for the latter.

[10]Halting conditions are represented explicitly by strategy rules that recommend a special halt action.

3.2 Eliciting Justifications and New Features from Experts

ASK's technique for eliciting justifications is based on a correspondence between the expert's justifications for choosing actions and the control features used by strategy rules. Justifications are facts about the current situation — instantiations of control features on relevant objects in working memory. The expert justifies the choice of an action by identifying those facts that are relevant to the decision. ASK provides a browsing interface (something like a friendly Lisp Inspector) for examining the state of working memory to find relevant facts. The expert searches for relevant objects (e.g., actions, hypotheses, goals) and the relevant control features that apply to them (cost of actions, confirming evidence for hypotheses), and indicates which facts serve as justifications. ASK guides the user by suggesting some objects that are potentially relevant. For example, all objects matched by successful strategy rules in the current cycle are automatically displayed in the relevant objects window.

The expert selects a justification by clicking on features of objects presented in a window. By clicking on the "cost" feature of the EKG object, the expert causes the statement "The cost of EKG is low" to appear in the justifications window. With a mouse-and-menu interface the expert manipulates the contents of the justifications window until ASK is satisfied (i.e., the set of justifications distinguishes EKG).

To help set the context and provide initial justifications, ASK applies heuristics to seed the justifications window. The set of seeded justifications serves as an explanation of the *system's* justifications for choosing the actions that it did.

If the available control features are inadequate, the expert can define a new feature. ASK provides a custom editor to help the expert define a new feature, using the context of the current object and existing features in the knowledge base to guide the process.

3.3 The Learning Cycle

ASK learns strategy rules from the expert's justifications for particular strategic decisions. The learning is structured in the dialog cycle, shown in Figure 3, consisting of the following operations.

1. **Credit assignment** determines what part of the system's knowledge needs to be changed, and how. The expert runs a case through the knowledge system, setting up an example where the system's performance (i.e., the actions recommended in a cycle) differs from the expert's. ASK analyzes the discrepancy between the expert's and system's choices, and then examines how the current set of strategy rules had matched. The output of credit assignment analysis is a specific learning objective, such as to specialize a particular selection rule so that it matches the expert's choice and does not match an action that the system suggested.
2. **Elicitation of justifications.** ASK helps the expert enter justifications, which it interprets as tests on control features, until it can formulate an expression that satisfies the learning objective.
3. **Formulating strategy rules.** ASK uses the justifications to form conditions on the left hand side of a new strategy rule. For example, if the expert says that the cost of the preferred action is relevant, ASK formulates a test such as `(= (cost ?action) low)`.

4. **Generalization.** By formulating new rules in terms of variables, ASK performs constant-to-variable induction. ASK also generalizes strategy rules with syntactic induction operators such as dropping conditions and extending intervals. ASK always asks the expert to verify the inductive generalizations it makes. It asks for verification of the constant-to-variable generalization by paraphrasing the rule, and queries other forms of generalization by showing their operational effects and by posing hypothetical variants on the training case.

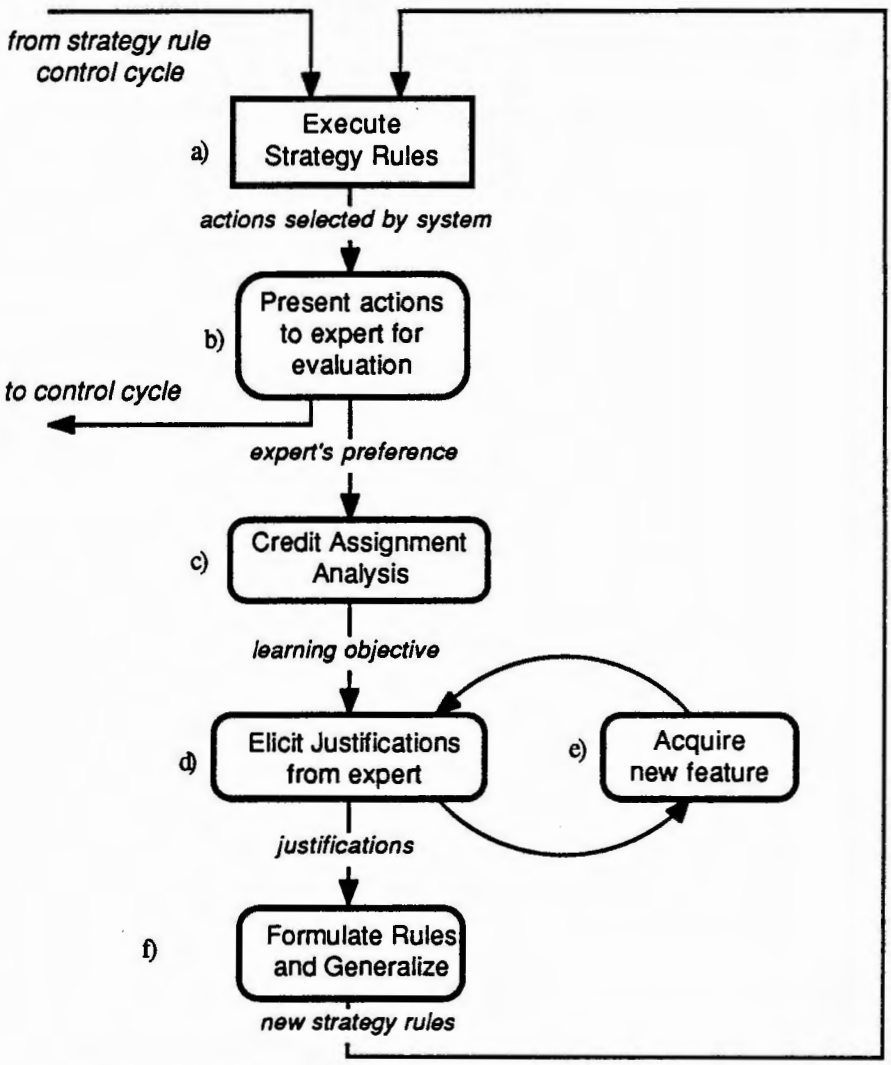


Figure 3. The ASK Knowledge Acquisition Dialog Cycle

4 WHY IT WORKS: SOURCES OF POWER AND LIMITATIONS

ASK is a synthesis of ideas and techniques from the AI literature. The approach demonstrates how several general ideas, such as explicit representation of control and knowledge elicitation in the context of performance, are combined to address a difficult problem: automating the acquisition of strategic knowledge. In addition, the ASK program

occupies an interesting point in a space of design parameters for knowledge acquisition tools, manifesting particular strengths and limitations. To evaluate the contribution of general ideas and specific design decisions, we have analyzed the system's *sources of power.* In this section we relate ASK's performance to some of its sources of power, and in the next section discuss some of the limitations that follow from the design.

4.1 Interactive Acquisition in a Performance Context

ASK was strongly influenced by the TEIRESIAS knowledge acquisition assistant (Davis, 1976). One of Davis' main claims was that a knowledge acquisition dialog can be structured around the interaction between an expert and an imperfect knowledge system. The reason that this is a useful technique is that it constrains an otherwise underconstrained activity; instead of asking "What should I know about this domain?" the assistant can ask "What went wrong with this case?" The acquisition in a performance context reduces both the scope of what is elicited and how it is operationalized. Like TEIRESIAS, ASK elicits a critique of a specific case, analyzes the discrepancy between the system's performance and the expert's preference, and uses knowledge about the contents of the existing knowledge base to suggest parts of the new rules (in the form of seeded justifications that turn into rule clauses). The performance context focuses the elicitation dialog around justifications, and the credit assignment analysis of the context tells ASK what the behavior the new rule must achieve (i.e., the operationality criterion). Like similar experiments in learning search strategy in SOAR (Golding, Rosenbloom, and Laird, 1987) and BB1 (Hayes-Roth and Hewett, 1985), ASK generalizes an evaluation of action choices from the internal state of the performance system and the teacher's critique of a case. In sum, acquiring knowledge in the context of cases of system performance guides both the human teacher and the learning assistant.

4.2 Specialization to a Task and Method

A second source of power for knowledge acquisition results from specializing in a particular performance task to which the acquired knowledge will be applied. Instead of providing the facilities for building *any* knowledge system, a task-specific knowledge acquisition tool offers a custom technique for acquiring particular kinds of knowledge, with well-specified roles in performing a specific class of tasks (McDermott, 1988). Furthermore, the associated task-specific architecture — the specialized knowledge representation and its method of being applied — serves as a partial design or model, leaving fewer degrees of freedom to be acquired and providing task-level representational primitives for the expert (Gruber and Cohen, 1987). The power of specialization has been realized by several knowledge acquisition assistants. Some specialize in the *problem-solving method,* the algorithm for applying domain knowledge to perform a task, such as MOLE's cover-and-differentiate method of heuristic classification (Eshelman 1988) and SALT's propose-and-revise method of iterative design (Marcus, 1988). Others can be described as specializing in a particular *kind of knowledge,* such as trait hierarchies acquired by ETS (Boose, 1986).[11] And some elicitation tools are specific to the *application task,* including the problem-solving method and the domain, such as OPAL's focus on skeletal refinement in managing cancer therapy protocols (Musen, Fagan, Combs, and Shortliffe, 1987; Musen, 1988).

[11]The knowledge acquired by the repertory grid tools like ETS is represented as weighted associations among elements and their traits. This knowledge may be used to perform various synthesis tasks, such as matching product requirements to products or multicriteria decision making (Boose, and Bradshaw 1987).

ASK makes a subtle cut across these dimensions of specialization. It is integrated with the MU architecture and the strategy rule representation of control, with the propose-filter-select method of applying strategic knowledge. It was used to refine the strategic knowledge for the task called prospective diagnosis in the chest pain domain, but it contains no specific knowledge about diagnostic tasks.[12] It *does* make a commitment to the class of tasks for which the reactive planning style of control is appropriate (see Section 5). It is clearly designed for the kind of knowledge we call strategic knowledge. But ASK differs from task-specific tools that use what McDermott (1988) has termed *role-limiting methods:* problem-solving methods that define clear roles for substantive knowledge and themselves incorporate task-specific control knowledge (MOLE and SALT are examples). ASK's method is much weaker, since it has to apply control knowledge that might normally be built into a role-limiting method by the engineer.

Thus ASK gains only some of the advantages of specialization. It does not reduce elicitation to form-filling, as is possible with OPAL, because it does not assume to know the relevant control features in advance. ASK's credit assignment algorithm can identify what kind of rules to acquire, and can suggest some possibly relevant objects and justifications, and this analysis is possible because of the restricted syntax and semantics of strategy rules. However, this is a weaker form of assistance than is possible when the method gives more specific roles for knowledge. MOLE's dynamic refinement strategy, for example, makes use of assumptions about how the links between symptoms and hypotheses are processed by the method to check for consistency and completeness (Eshelman, 1987). Nonetheless, like all task-specific tools, ASK gains leverage by restricting what can be elicited from experts and simplifying how the elicited knowledge is put into operational form. The only way to enter strategic knowledge is to list justifications for particular strategic choices; justifications are always well defined for the system and explainable to the user; and justifications can be mechanically transformed into operational pattern-matching clauses for rules.

4.3 Explicit, Declarative Representation of Control

Several researchers have cited the advantages of representing control knowledge explicitly, for explanation, software engineering, and knowledge acquisition purposes (Clancey, 1983b; Gruber and Cohen, 1987; Hayes-Roth, 1985; Neches, Swartout, and Moore, 1985; Silver, 1986). If the control knowledge of a program is embedded in a procedure, or worse yet, implicit in priorities and rule orderings, then it is difficult for people and programs to analyze, explain, and maintain it. Because its representation for control is explicit rules, ASK can reason about their behavior (for credit assignment), and can make use of strategic knowledge in a form that experts can provide (i.e., justifications). The idea of representing control knowledge as explicit rules was introduced by Davis (Davis, 1980), and was developed as a representation of strategy by Clancey (1984; Clancey and Bock, 1988). The productions that create preferences in SOAR (Laird, Newell, and Rosenbloom, 1987) formulate control decisions with rules in a manner similar to ASK. In every case, the rule formalism is used because it makes the knowledge about what to do next *explicit*, so that it can be reasoned about by the system and understood by people. ASK shows that making control knowledge explicit makes it possible to acquire it from experts.

---

[12]The general strategy for prospective diagnosis can be found in a small set of strategy rules that were in the knowledge base prior to the example dialog in this paper. A subsequent experiment established that the general strategy, too, can be acquired from scratch using ASK.

4.5 Fine-grained Control and Local Decisions

In our experience, experts in a task have more difficulty devising a general, operational description of their strategy than describing what they actually *do* in specific cases. ASK shows that strategic knowledge can be acquired from experts if it is elicited in the context of specific choices among actions and then generalized. This is possible because strategy rules model iterative, local decisions about actions; a global strategy emerges from a series of strategic decisions. Similarly, SALT allows designers to provide knowledge that is relatively easy to specify — details about constraints among individual pieces — and offers help for putting the pieces together (Marcus, 1988). It too is based on a method of making local decisions (with backtracking), proposing one plan extension or fixing a single constraint at a time.

The technique of acquiring small-grained decisions is also a limitation, since it does not encourage the user to develop an explicit global strategy. If a simple procedure does in fact adequately capture the domain strategy, then ASK's rule formalism is inappropriate. The grain-size of the control decisions is a design parameter that illustrates the tradeoff between acquirability and expressiveness discussed in the conclusions.

4.6 Generalizable Formalism

Strategy rules can represent levels of generality of strategic knowledge. The more specific strategy rules are more constraining — they choose fewer actions for more reasons, and take precedence over more general rules. Furthermore, the generality relationship among rules can be determined syntactically — it is reflected in their form. A knowledge representation that can capture levels of generality in its form is generalizable. The generalizability of strategy rules facilitates knowledge acquisition in two ways. First, since a developing system can fall back on general strategy, it can be run with an incomplete base of strategic knowledge. This allows the knowledge engineer to write a few general strategy rules and the expert to "fine-tune" the strategy by specializing them. Second, the correspondence between the syntactic structure and the semantics of generality in strategy rules allows ASK to use simple syntactic induction operators for learning.

The generalizability of the representation, and in particular, the presence of generality relation in the structure of the representation language, has long been recognized as an important bias for machine learning (Mitchell 1982). The hierarchical structure of the ACCORD control language is a principle source of power of a system called WATCH that generalizes search strategies from examples (Hayes-Roth, Garvey, Johnson, and Hewett, 1987). Both ASK's and WATCH's representations of strategy allow for levels of generality, and as a consequence both systems can use syntactic induction to find generalizations.

4.7 Reformulating Strategy as Classification

Strategy is naturally thought of in procedural terms, because that is how programs know what do to next. But representing strategy as procedures complicates induction. The problem of learning sequences from examples, much less arbitrary procedures with internal state, is very hard (Dietterich and Michalski, 1986). On the other hand, the problem of learning classification rules is well understood (Michalski, Carbonell, and Mitchell, 1983). ASK reformulates the problem of "how to act" as mappings between situations and equivalence classes of actions — that is, as classification rules. As a consequence, ASK can use straightforward induction operators. Reformulating control as classification is an old

and powerful technique. Several systems for learning search control knowledge represent control with productions (e.g., LP (Silver, 1986) , PRODIGY/EBL (Minton ,and Carbonell, 1987) and SOAR (Laird, Newell, and Rosenbloom, 1987). Waterman formalized a production system for the explicit purpose of learning strategy; in his Poker Player, productions mapped heuristic descriptions of the state of the game to actions such as "bet" (Waterman, 1970).[13]

4.8 Integrating Machine Learning and Interactive Knowledge Acquisition

ASK integrates interactive elicitation with induction. It learns with a human in the loop, by asking for experts' knowledge and operationalizing and generalizing what it is told. The human and program play complementary roles in the learning process. The expert selects examples as a teacher, and example selection is a way to get inductive learning algorithms to learn useful concepts (Winston, 1975; Rissland, 1987). The knowledge engineer and expert develop and extend the terms in the description language, which is another source of strong bias for learning systems (Utgoff, 1986). Because the expert is part of the learning process, available to define terms when needed, ASK is less susceptible to the new term problem — that learning systems cannot learn what they cannot represent. Further, ASK can "learn at the knowledge level" (Dietterich, 1986), since the expert imparts new facts to the system. While the expert serves as critic, the role of the learning element is played by ASK: the program performs credit assignment, operationalization, and generalization. The human-machine synergy that results from integrating elicitation and learning techniques is a source of power for several knowledge acquisition systems (Bareiss, 1987; Boose and Bradshaw, 1987; Eshelman, 1987; Haas and Hendrix, 1983; Morik, 1987) and some classic machine learning research (Lenat, 1983).

5 SCOPE OF APPLICABILITY

The problems to which ASK can be applied are those for which expert strategy is essential to the performance task and for which the strategy rule representation is adequate. The latter condition is, of course, shorthand for the familiar problem of matching methods to tasks. So to describe ASK's applicability we must discuss characteristics of the actions, goals, and features of a task in terms of their match to ASK's method of applying strategic knowledge. The major characteristics are:

- *Actions can be selected one at a time (as opposed to planning sequences of strongly interacting steps).* A positive example is reactive planning for robots, where uncertainty about the world and real-time constraints necessitate acting without projection. A negative example is prescribing a set of antibiotics to cover an infection, because the actions (drugs) interact and must be planned as a set. The strategy rule representation is not rich enough to capture the kind of drug-selection strategy such as MYCIN's, which had to be implemented in Lisp (Clancey, 1984b).
- *Actions can be modelled as primitive steps, enumerated at run time.* A positive example is selecting legal cases for argument; the choice of a case can be considered on its own merits, relatively independent of other cases. A negative example is the management of cancer treatment plans, the domain of ONCOCIN (Shortliffe, Scott, Bischoff, van Melle, and Jacobs, 1981). Experts in cancer treatment apply strategic knowledge to generate *protocols*: long, detailed, branching plans (sequences of

[13]This technique is closely related to the previous two sources of power. Reformulating strategy as classification facilitates induction because the classification knowledge is represented in an easily generalizable form, such as conjunctions of features, and this depends on an explicit, declarative representation of control!

therapeutic actions). We found that the justifications for choosing the next action in protocols were often inter-action dependencies of the form "because I just did action-X and it had this result," rather than "because action-Y has the appropriate properties for this patient in this situation." It seems that many of the features of the situation that determine which action should go next are really descriptions of immediate past history. Although one could certainly refer to the action history as a justification and generate strategy rules, they would not generalize well, because the terms would be constants (specific actions). This is reflected in a difference in problem solving methods; ONCOCIN's designers chose a skeletal plan refinement method, while ASK uses the reactive style of planning.

- *Local action-selection criteria can avoid global pitfalls.* Computer players of many adversarial games are often based on static evaluation of position. Their strategy is to repeatedly choose actions that score best on the evaluation function. This works if the evaluation function embodies action-selection criteria that do not lead to global pitfalls. ASK could be used to acquire the kind of strategy learned by Waterman's poker player — mappings from descriptions of board state and the opponent to actions. This strategy will win if the descriptions are usefully predictive; if what looks good locally does not turn out to be a trap. A borderline negative example is chess, where strategy is often played out over several moves and evaluation functions are prone to horizon effects. If the right features can be found, strategy rules can map them to actions and ASK can acquire them. If the features that the expert can invent lead to pitfalls, then acquiring rules that use these features will not produce a globally optimal strategy.
- *An optimal decision among actions is not required or possible for every choice of actions.* Our chest pain application is a positive example: most of the evidence-gathering questions, tests, and therapies are chosen with relatively simple measures of utility. The data and need to elicit probabilities and numeric estimates of utility for every possible combination of actions is not present in practice. However, an excellent negative example in the very same domain is the *last* strategic decision that is made: deciding whether to perform angiography and consequently open heart surgery. The decision has been successfully modelled using the techniques of decision analysis (Pauker and Kassirer, 1981).

6 MAJOR LIMITATIONS

Aside from restrictions of what can be acquired, ASK is limited by some of the assumptions made by its acquisition techniques.

6.1 Multiple Experts

One of the traditional motivations for separating strategic and substantive knowledge is the hope that the the strategy could be used in several systems within a domain. By acquiring a single expert's strategy, we may be biasing the resulting system to use an idiosyncratic approach. If two or more experts work on the same knowledge system, their strategic knowledge may differ significantly. ASK has no mechanism to merge the recommendations of several experts.

6.2 The Substantive Knowledge must be Acquired and Correct

ASK builds upon an existing substantive knowledge base, relying on the objects and features already defined. It is assumed that the substantive knowledge can be acquired, possibly with a tool for heuristic classification, such as MOLE. A more serious problem is the assumption that the substantive knowledge is correct. ASK's credit assignment algorithm cannot handle bugs in the substantive knowledge base, and they are certain to occur. The user must be able to recognize this problem when it occurs.[14]

6.3 Dependence on Knowledge Engineers

Although ASK provides a helpful interface for defining new control features, some require programming to implement. These include functions which compute dynamic evidential relationships between actions and hypotheses; they must be implemented by knowledge engineers. And of course ASK cannot be used in isolation to implement a knowledge system; its speciality is the strategic component. Yet like ETS, ASK can serve the purpose of identifying features that are relevant to an expert's domain, without the intervention of a knowledge engineer. When a sufficient set of control features is in place, ASK can help the expert iteratively refine and extend a strategy based on those features.

7 CONCLUSION

ASK's design is a tightly coupled integration of an operational representation, an elicitation technique, and a learning method. The representation is constrained by limitations on what can be elicited from experts. The learning method depends on the expert for examples and bias, and it relies on properties of the representation for generalization. Overall, the sources of power and the limitations we have discussed result from a design which balances the expressive power of the representation with the objectives for automating the knowledge acquisition dialog.

ACKNOWLEDGEMENTS

This paper reports on my Ph.D. dissertation with Paul Cohen at the University of Massachusetts. The research was made possible by his insightful collaboration and guidance. I am grateful to my colleagues at the EKSL for inspiration and feedback and to Ken Ward for his technical assistance. Thanks to Kevin Ashley, Jim Bennett, Sabine Bergler, David Day, Edmund Durfee, Adele Howe, Mark Musen, and Nancy Wogrin for thoughtful reviews.

REFERENCES

Bareiss, E. R., Porter, B. W., and Wier, C. C. (1987). Protos: an exemplar-based learning apprentice. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October. To appear in the *International Journal of Man-Machine Studies*.

---

[14] A surprising side-effect of ASK's sensitivity to the substantive knowledge base is its usefulness as a testing vehicle for knowledge engineers. A bug in a control feature, for example, might cause the "wrong" strategy rule to fire or fail to fire, producing a bad choice of action. If that control feature is given as the justification for choosing the action, ASK will complain that it needs more justifications for distinguishing an action that should have been distinguished — but had a faulty control feature. The ODYSSEUS learning apprentice exploits this sensitivity of metarules to incompleteness in substantive knowledge reflected in the metarelations; it assumes correct strategic knowledge and uses it as a model to acquire substantive knowledge (Wilkins, Clancey, and Buchanan, 1987).

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

Boose, J. H. and Bradshaw, J. M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for expert systems. *International Journal of Man-Machine Studies*, 26(1):21–25.

Chapman, D. and Agre, P. E. (1987). Abstract reasoning as emergent from concrete activity. In M. P. Georgeff and A. L. Lansky (Eds.), *Reasoning About Actions and Plans, Proceedings of the 1986 Workshop at Timberline, Oregon*, pages 411–424.

Clancey, W. J. (1983a). The epistemology of a rule-based expert system — A framework for explanation. *Artificial Intelligence* 20:215–251.

Clancey, W. J. (1983b). The advantages of abstract control knowledge in expert system design. In *Proceedings of the Third National Conference on Artificial Intelligence*, pages 74–78, Washington, D. C., August.

Clancey, W. J. (1984a). Details of the revised therapy algorithm. In B. G. Buchanan and E. H. Shortliffe (Eds.), *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*, Addison-Wesley, Reading, MA, pages 133–146.

Clancey, W. J. (1984b). Acquiring, representing, and evaluating a competence model of diagnosis. KSL Memo 84-2, Stanford University, February. To appear in Chi, Glaser, and Farr (Eds.), *Contributions to the Nature of Expertise*, in preparation.

Clancey, W. J. and Bock, C. (1988). Representing control knowledge as abstract tasks and metarules. In L. Bolc and M. Coombs (Eds.), *Expert System Applications*, Springer-Verlag, 1988, pages 1-78.

Cohen, P. R., Day, D. S., Delisio, J., Greenberg, M., Kjeldsen, R., Suthers, D., and Berman, P. (1987). Management of uncertainty in medicine. *International Journal of Approximate Reasoning*, 1(1):103–116.

Cohen, P. R., Greenberg, M., and Delisio, J. (1987). MU: A development environment for prospective reasoning systems. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 783–788, Seattle, Washington, July.

Davis, R. (1976). *Applications of meta-level knowledge to the construction, maintenance, and use of large knowledge bases*. PhD thesis, Computer Science Department, Stanford University. Reprinted in R. Davis and D. B. Lenat (Eds.), *Knowledge-Based Systems in Artificial Intelligence*, McGraw-Hill, New York, 1982.

Davis, R. (1980). Meta-rules: Reasoning about control. *Artificial Intelligence*, 15(3):179–222.

Dietterich, T. G., London, B., Clarkson, K., and Dromey, G. (1982). Learning and inductive inference. In P. R. Cohen and E. Feigenbaum (Eds.), *The Handbook of Artificial Intelligence, Volume III*, William Kaufmann, Inc.

Dietterich, T. G. (1986). Learning at the knowledge level. *Machine Learning*, 1(3):287–316.

Dietterich, T. G. and Michalski, R. S. (1986). Learning to predict sequences. In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach, Volume II*, pages 63–106, Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

Erman, L. D., Scott, A. C., and London, P. E. (1984). Separating and integrating control in a rule-based tool. In *Proceedings of the IEEE Workshop on Principles of Knowledge-base Systems*, pages 37–43, Denver, Colorado, December.

Eshelman, L. (1987). MOLE: A knowledge acquisition tool that buries certainty factors. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October. To appear in the *International Journal of Man-Machine Studies*.

Eshelman, L. (1988). MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems*, pages 37–80, Kluwer Academic Publishers, Boston.

Eshelman, L., Ehret, J., McDermott, J., and Tan, M. (1987). MOLE: A tenacious knowledge acquisition tool. *International Journal of Man-Machine Studies*, 26(1):41–54.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 202–206, Seattle, Washington.

Gruber, T. R. (1987). Acquiring strategic knowledge from experts. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October. To appear in the *International Journal of Man-Machine Studies*.

Gruber, T. R. and Cohen, P. R. (1987). Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *International Journal of Man-Machine Studies*, 26(2):143–159.

Golding, A., Rosenbloom, P. S., and Laird, J. E. (1987). Learning general search control from outside guidance. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 334–337, Milan, Italy, August.

Haas, N. and Hendrix, G. G. (1983). Learning by being told: Acquiring knowledge for information management. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, pages 405–428, Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

Hayes-Roth, B. (1985). A blackboard architecture for control. *Artificial Intelligence*, 26:251–321.

Hayes-Roth, B., Garvey, A., Johnson, M., and Hewett, M. (1987). *A Layered Environment for Reasoning About Action*. Technical Report KSL 86–38, Computer Science Department, Stanford University, April.

Hayes-Roth, B., and Hewett, M. (1985). *Learning control heuristics in a blackboard environment*. Technical Report HPP-85–2, Computer Science Department, Stanford University.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64.

Lenat, D. B. (1983). EURISKO: a program that learns new heuristics and domain concepts. *Artificial Intelligence*, 21:61–98.

Marcus, S. (1987). Taking backtracking with a grain of SALT. *International Journal of Man-Machine Studies*, 26(4):383–398.

Marcus, S. (1988). SALT: A knowledge acquisition tool for propose-and- refine systems. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems*, pages 81–121, Kluwer Academic Publishers, Boston.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus (Ed.), *Automating Knowledge Acquisition for Expert Systems*, pages 225–256, Kluwer Academic Publishers, Boston.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (1983). *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann Publishers, Inc., Palo Alto, CA.

Minton, S. and Carbonell, J. G. (1987). Strategies for learning search control rules: An explanation-based approach. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

Morik, K. (1987). Acquiring domain models. *International Journal of Man-Machine Studies*, 26(1):93–104.

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

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.

Neches, R., Swartout, W., and Moore, J. (1985). Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11(11):1337–1351.

Pauker, S. G. and Kassirer, J. P. (1981). Clinical decision analysis by personal computer. *Archives of Internal Medicine*, 141:1831–1837.

Rissland, E. L. (1987). The problem of intelligent example selection. In *Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-based Systems Workshop*, Banff, Canada, October. To appear in the *International Journal of Man-Machine Studies.*

Shortliffe, E. H., Scott, A. C., Bischoff, M. B., van Melle, W., and Jacobs, C. D. (1981). ONCOCIN: An expert system for oncology protocol management. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 876–881, Vancouver, British Columbia, August.

Silver, B. (1986). *Meta-level Inference: Representing and Learning Control Information in Artificial Intelligence.* North-Holland, New York, NY.

Utgoff, P. (1986). *Machine Learning of Inductive Bias.* Kluwer Academic Publishers, Boston.

Waterman, D. A. (1970). Generalization learning techniques for automating the learning of heuristics. *Artificial Intelligence*, 1:121–170.

Wilkins, D. C., Clancey, W. J., and Buchanan, B. G. (1987). Knowledge base refinement by monitoring abstract control knowledge. *International Journal of Man-Machine Studies*, 26:281–293.

Winston, P. H. (1985). *Learning structural descriptions from examples.* McGraw Hill, New York.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Learning Importance of Concepts: Construction of Representative Network

Ken-ichi Handa Shun Ishizaki

ElectroTechnical Laboratory
Umezono 1-1-4, Tsukuba, JAPAN, 305
ARPA: handa%etl.jp@relay.cs.net

Abstract

A method for learning importance of concepts and for constructing representative network through the learnt importance is proposed. Calculation of importance is done by propagating appropriate values over a semantic network for a piece of knowledge and the importance is refined incrementally through processing much more knowledge. The method has been verified on a knowledge base of complicated hierarchy and the examination has shown the effectiveness of this method for further knowledge acquisition process or knowledge base management.

1 INTRODUCTION

Automatic concept learning is a big topic in the field of machine learning. Considering variety and complexity of real world's knowledge, a concept learning system should deal with a larger amount of complex input data. Usefulness of generalization-based method for such a system has been shown in [Quinlan 79,Lebowitz 83]. One of the advantages of their systems is that they do not require a priori knowledge about what concepts should be learned. Another is that there is no need for organizing input data carefully so as to control learning process. On the other hand, these systems have such a large space of knowledge that search for examples to be compared for creating new concepts can hardly be conducted effectively.

The similar ineffective search also occurs in a system which learns concepts by asking questions [Sammut 83]. With this system, a teaching load increases on the part of a teacher because he must answer an unnecessarily large number of questions when the system is faced with a large amount of knowledge data.

For a knowledge base system, an effective guideline to indicate from where to start the reorganization of knowledge base is required. Moreover, with such guideline, the system can provide a good man-machine interface to help human understand a large volume of knowledge data rapidly.

Looking at these problems, we realize it is necessary for a knowledge processing system to understand a degree of importance of each concept. We discuss in this paper how the system learns such importance as it acquires new knowledge. Articles of newspapers are used for the data to be input to the system. We also propose a method for creating a *representative network* through the importance learnt. The methods have the following features.

- Concepts acquired from newspapers have complicated hierarchies.
- Degree of importance is determined from hierarchies and lateral relations among concepts.
- The system learns importance of general knowledge on a incremental basis as newspaper articles are processed.

2 OUTLINE OF IMPORTANCE LEARNING PROCESS

The knowledge acquired from newspapers is classified into two categories. One is *frame knowledge* (generally acquired over many articles) and another is *specific knowledge* (knowledge appearing only in specific articles). This classification enables the system to digest each article in a realistic time even when the amount of knowledge in the system increases.

The process of learning the degree of importance of concepts are shown below.

1. **[Representing knowledge in semantic network]**
   Represent knowledge acquired from one article in semantic network. Concepts in the network are classified into either *frame knowledge* or *specific knowledge* by human.

2. **[Calculating importance value]**
   A value assigned to each concept of both *frame knowledge* stored so far and *specific knowledge* of the current article is propagated on the network just like an activation value of PDP (Parallel Distributed Processing) models. We consider the converged value of the concept represents the degree of importance of that concept and call this the *importance value* . We can set a priori bias to the initial value of a concept of *frame knowledge* so as to control the propagation.

3. **[Identifying representative network]**
   A network of those concepts having higher *importance values* is identified as a *representative network* . The *representative network* is stored in the main memory so that the system can access quickly while the rest of *specific knowledge* network are stored in the secondary memory.

4. **[Learning *importance value* of frame knowledge]**
   Importance values of *frame knowledge* is normalized according to the size of the network and memorized for use in the next calculation.

The more articles are processed, the more the accuracy of the *importance value* of the *frame knowledge* improves. And so does the validity of the *representative network*. In addition, we can expect the effective use of the resource and improvement of the performance because only the important part of *specific knowledge* network is stored in a place where the system can access quickly.

Now, let us discuss each step of the process in detail.

3 REPRESENTING KNOWLEDGE IN SEMANTIC NETWORK

We chose for input data newspapers articles on new models of workstations and translated them into semantic network-based knowledge representation language IXL [Handa 86]. This translation is done manually but also automatically as much as possible by referring to a comparison table prepared for expressions used in newspapers and corresponding IXL statements.

IXL is a very simple langage for creating semantic network, the main command of which is `unit` and `relation`. The examples of these commands are shown in Figure 1. `unit` command creates a simple structure, *unit*, of two concepts connected by one of links predefined in IXL, and `relation` command creates more complicated structure, *relation*, by connecting seven related *units* [Handa 86]. `isa` and `ins` links respectively represent is_a and instance_of hierarchy. `s:?` and `d:?` links shows how a concept concerns with a relationship.

The `unit` in the figure represents that all the instances of *EWS* are also instances of *computer*. The `relation` in the figure represents the following more complicated relationship.

> *YHP* itself *sells* all the instances of *350CHX*. In addition, all the instances of *company* necessarily *sell* some instances of *products* and all the instances of *products* are *sold* by a certain instance of *company*.

We call *sell/isa/12*, whose name is generated automatically, the *anchor* of the *relation*. because it is the key concept which combine three related concepts, *350CHX*, *YHP*, and *sell*.

```
unit(isa,'EWS',computer).   relation(sell:direct,company:1,products:some).
                            relation(sell,'YHP':self,'350CHX':all).
```

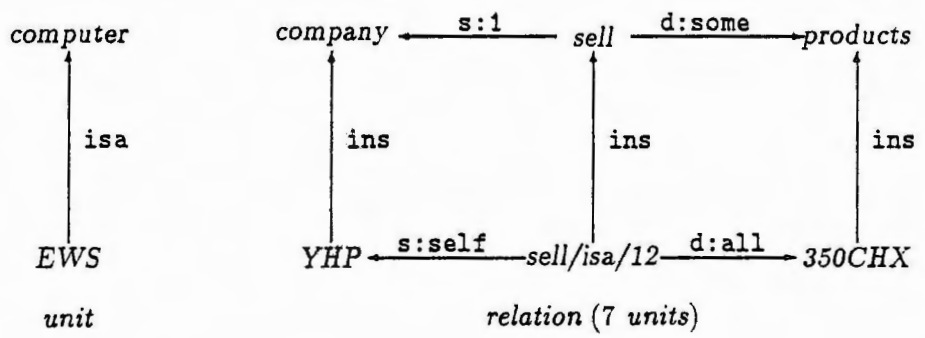


Figure 1: structure of *unit* and *relation*

IXL statements:

```
unit(isa, president, human).
relation(president_is:direct, company:some, president:1).
relation(sell:direct, company:1, products:some).
...
```

Meaning:

"*president* is a subclass of *human*. *president_is* is a relation between instances of *company* and *president*. *sell* is a relation between *company* and *products*, ..."

Figure 2: Example of IXL statements for *frame knowledge*

IXL statements:

```
YHP = 'Yokogawa_Hewlett_Packard',
unit(isa, NewHP, 'EWS'), unit(ako, ['350CHX','330CHX'], NewHP),
relation(sell, YHP:self, NewHP:all),
relation(president_is, YHP:self, 'Sasaoka Kenzo':self),
...
```

The translated article (source language: Japanese):

"Yokogawa Hewlett Packard put new engineering work stations 'HP9000 model 350CHX' and '330CHX' on the market, ... and the president is Mr. Sasaoka Kenzou, ..."

Figure 3: Example of IXL statements for *specific knowledge*

At present, classification of knowledge into *frame knowledge* and *specific knowledge* is done by human. We also gave the system in advance such basic element of knowledge as relationship between numerals and units and common concepts such as president and company — concepts not obtainable from newspapers but necessary for understanding article.

Figure 2 is IXL statements for a part of *frame knowledge* given in advance, and Figure 3 is those for *specific knowledge*, the original article and the whole IXL statements for which are in Appendix A and B respectively.

We have already translated 70 articles into IXL, and the size of the resulting semantic network is in Table 1.

type	size of network	
frame knowledge (total)	172 links	106 nodes
specific knowledge (average)	240 links	111 nodes

Table 1: Size of Network

4 CALCULATING IMPORTANCE VALUE

As in machine learning, much effort has also made to extract important concepts for the purpose of creating an abstract of sentences in the field of natural language processing. The PDP-based method for abstraction proposed in [Hasida 87] suits with semantic network knowledge base because the method uses information how concepts are connected semantically, unlike other methods that use syntactical information of a source sentence. The following reflects a brief picture of this method, and explains how it is integrated into our system.

4.1 Signal Propagation On Semantic Network

Let us consider a network in which two very closely related concepts are connected directly by a link. The criteria of importance of concepts we can take into account on this network are:

- A concept connected to an important concept is also important,
- A concept connected to many other concepts is important.

These two criteria are simultaneously accommodated by such a single PDP-based method as each nodes propagates its own activation value via links according to the connection coefficients of these links. In this situation, the activation value of each node at the point of propagation convergence can be regarded as the *importance value* of that node. This process is mathematically formulated as the followings.

Here we adopt an approximation where time is quantized and the signal propagation is synchronized. Let all the relevant nodes be numbered sequentially from 1 to n; $x_{t,i}$ be the activation value of the node i at time t; $a_{i,j}$ be the connection coefficient of the link between nodes i and j; c_i be the constant activation value poured into the node i, and we have:

$$\vec{X}_0 = \vec{C}, \tag{1}$$

$$\vec{X}_{t+1} = A\vec{X}_t + \vec{C}\ (t = 0, 1, 2, \ldots), \tag{2}$$

where,

$$\vec{X}_t = \begin{pmatrix} x_{t,1} \\ \vdots \\ x_{t,n} \end{pmatrix}, A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix}, \vec{C} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix}.$$

In case all eigenvalues of A are less than 1, $\vec{X}_{t\to\infty}$ converges. To guarantee this convergence, we defined $a_{i,j}$ (an element of A) as follows:

$$a_{i,j} = \frac{l_{i,j}}{\mathrm{MAX}_{k=1}^{n}(d_k)} \tag{3}$$

where, $l_{i,j}$ is an element of adjacent matrix L, $l_{i,i} = 1$, $d_k = \sum_{j=1}^{n} l_{k,j}$).

4.2 Applying Signal Propagation Method To A Network

Now we show how the method described above are applied to the semantic network we gained from a newspaper article. Let us assume that any learning have not yet be done (i.e. the initial stage of the learning process).

The example is the article shown in Appendix A. The network for this article consists of *frame knowledge* (62 nodes, 83 links) and *specific knowledge* (128 nodes, 248 links). The IXL program to create this network is shown in Appendix B. Since the matrix A in Expression (2) is sparse (it is because most nodes have only 2 or 3 links), we can get a limit of $\vec{X}_t$ within a realistic time just by applying Expression (2) repeatedly (iterative method). For simplicity, we assume the value of each element of $\vec{C}$ is 1 (remember that this is an initial stage and that the system, therefore, have no information on importance of each concept).

Table 2 shows top 20 nodes of higher *importance value* . A node of such a strange name as *XXX/isa/nnn* means that it has no name of its own and is a subclass of *XXX*. *nnn* is the identification number given automatically by the system. For instance, *EWS/isa/11* is a subclass of EWS and referred to in the article as "the new models" and *EWS/isa/25* is a concept corresponding to *Yokogawa_Hewlett_Packard*'s old models.

frame knowledge		*specific knowledge*	
nodes (concepts)	*importance value*	nodes (concepts)	*importance value*
has_part	2.019	EWS/isa/11	2.377
component	1.981	Yokogawa_Hewlett_Packard	1.598
computer	1.776	330CHX	1.456
volume	1.622	350CHX	1.456
storage_device	1.487	graphics_processing/isa/27	1.425
products	1.462	speed_is/isa/32	1.391
body_part	1.457	MC68020	1.383
speed_is	1.397	main_storage/isa/87	1.371
drawing_speed	1.391	main_storage/isa/80	1.371
operation	1.386	HP-UX	1.362
process_is	1.338	color_display/isa/90	1.359
workstation	1.335	color_display/isa/83	1.359
processing_speed	1.329	LAN/isa/72	1.352
CPU	1.311	EWS/isa/25	1.339
OS	1.305	has_part/isa/71	1.306
display	1.299	has_part/isa/76	1.306
has_part/isa/5	1.285	has_part/isa/49	1.302
sell	1.282	has_part/isa/54	1.302
capacity_is	1.278	has_part/isa/78	1.302
...	...	...	...

Table 2: Example of *importance value* (top 20 concepts)

Almost all of the important concepts are listed in the table for *specific knowledge* , which we can examine by comparing the list and the original article (in Appendix A). The concept *speed_is/isa/32* shows that the description about the speed of *graphics_processing* is also the important part of the article.

On the other hand, from the important nodes of the *frame knowledge* , we can understand the tendency of information contained in the article. Such tendency is, for instance, that structural concepts (i.e. what kind of *components computer* has) and speed of some *operations* hold a significant place.

It must be noticed that the purpose of calculating *importance value* is not to figure it accurately but to give a guideline for such processes as knowledge acquisition and knowledge base management, which are coming later. All we need is a rough outline of an article.

4.3 Giving a priori bias on importance value

Which concepts are important in an article is, of course, heavily depends on what kind of information we are interested in. A method how to inform the system of what we are especially interested in is described below.

We introduce here a priori bias on an *importance value* of certain concept. As you see from the Expression (2), a limit of each element of $\vec{T}_t$ is higher than each corresponding element of $\vec{C}$. So, the simplest way is to assign a higher value to the corresponding constant activation value.

For example, Table 2 does not contain a full information about *HP-UX*. Supposing we were very interested in operating systems, all we have to do is to give the following command (Horn Clause of Prolog):

```
nodev(3.0, 'OS'),
```

which assigns 3.0 to c_i of *OS*. This biasing accordingly changes the *importance value* of *specific knowledge* as shown in the left hand part of Table 3. We still have to find out that the relation between *HP-UX* and *based_on* is *based_on*, which, however, is not difficult after the concept *UNIX* appearing.

In addition, values of $l_{i,j}$ don't have to be 1 should the corresponding link really exist. Even if a higher value is assigned to $l_{i,j}$, the convergence is guaranteed as long as Expression (3) holds. It means that we can also put bias on a connection coefficient of a certain link.

```
connv(2.0, r, a, b)
```

assigns 2.0 to the element of matrix L which corresponds to the link `r` between node `a` and `b`.

The right hand part of Table 3 is a resulting *importance value* when the following command is given to the system.

```
assert((connv(Link,X,Y,2.0) :-
        (Link=src:_; Link=dst:_), isa(X,speed_is)))
```

The purpose of this command is to combine a computer and its speed more tightly. The effect of bias is apparent because three more concepts related to the relationship *speed_is* appear in the

bias on $\vec{C}$		bias on A	
nodes (concepts)	*importance value*	nodes (concepts)	*importance value*
EWS/isa/7	2.378	EWS/isa/7	2.627
Yokogawa_Hewlett_Packard	1.598	Yokogawa_Hewlett_Packard	1.599
HP-UX	1.467	speed_is/isa/30	1.537
330CHX	1.456	speed_is/isa/30/isa/33	1.476
350CHX	1.456	drawing_speed/isa/53	1.476
graphics_processing/isa/25	1.425	330CHX	1.469
UNIX	1.423	350CHX	1.469
speed_is/isa/30	1.391	processing_speed/isa/48	1.462
MC68020	1.384	graphics_processing/isa/25	1.427
main_storage/isa/85	1.376	EWS/isa/23	1.413
...	...	...	...

Table 3: Changes of *importance value* by biasing (top 10 concepts)

table. Although these concepts themselves do not show their values but they are the *anchors* of the relationships, we know from them that there exist at least three pieces of information about the speed of the computers in the article. We believe, as mentioned above, this kind of information is useful for the later knowledge base processing.

We note here that we do not intend to advocate this PDP-based method simulates any cognitive models of human. We have adopted the method only because the importance of concepts are determined according to the criteria mentioned earlier in this section.

5 REPRESENTATIVE NETWORK

In this section, we describe how to extract a summary part from a network of *specific knowledge* using *importance value* s obtained through the preceding steps.

We must, at first, select concepts of higher *importance value* . The threshold should be decided according to an amount of system resources (e.g. memory capacity, CPU power, and etc.), or should be left to the users' decision. We select top 20 concepts (about top 15 %)of higher *importance value* out of *specific knowledge* for this time.

Since a meaning of a concept is defined in relation to the other concepts, not only an important concepts but also a link is necessary for not losing information of an original article. Although most concepts in *specific knowledge* have links to a concept in *frame knowledge* , these links do not suffice for adequate representation. Let us consider the situation that *HP-UX* and *UNIX* are selected for important concepts as in Table 3 and both have a link to *OS*. It is apparent, however, that we need another links which show how they are related to each other.

So, we use the following algorithm to extract only a necessary part from a whole network. We call the extracted part as *representative network* of the original article.

1. Activate all links which connect two important nodes directly. Now we have several networks which consist of important nodes connected directly or indirectly by activated links. We call these networks *islands*.

2. Define the *importance value* of *island* equals to the highest *importance value* of nodes in the *island*.

3. Choose two *islands* of highest *importance value* and find the shortest paths (i.e. path with least unimportant nodes) to connect two nodes which belong to those respective *islands*. We call these paths *bridges*.

4. Sum up *importance value* of nodes on each *bridge* (including nodes on the ends) and assign the summed up value to the *bridge* as the *importance value* . Activate all links of the

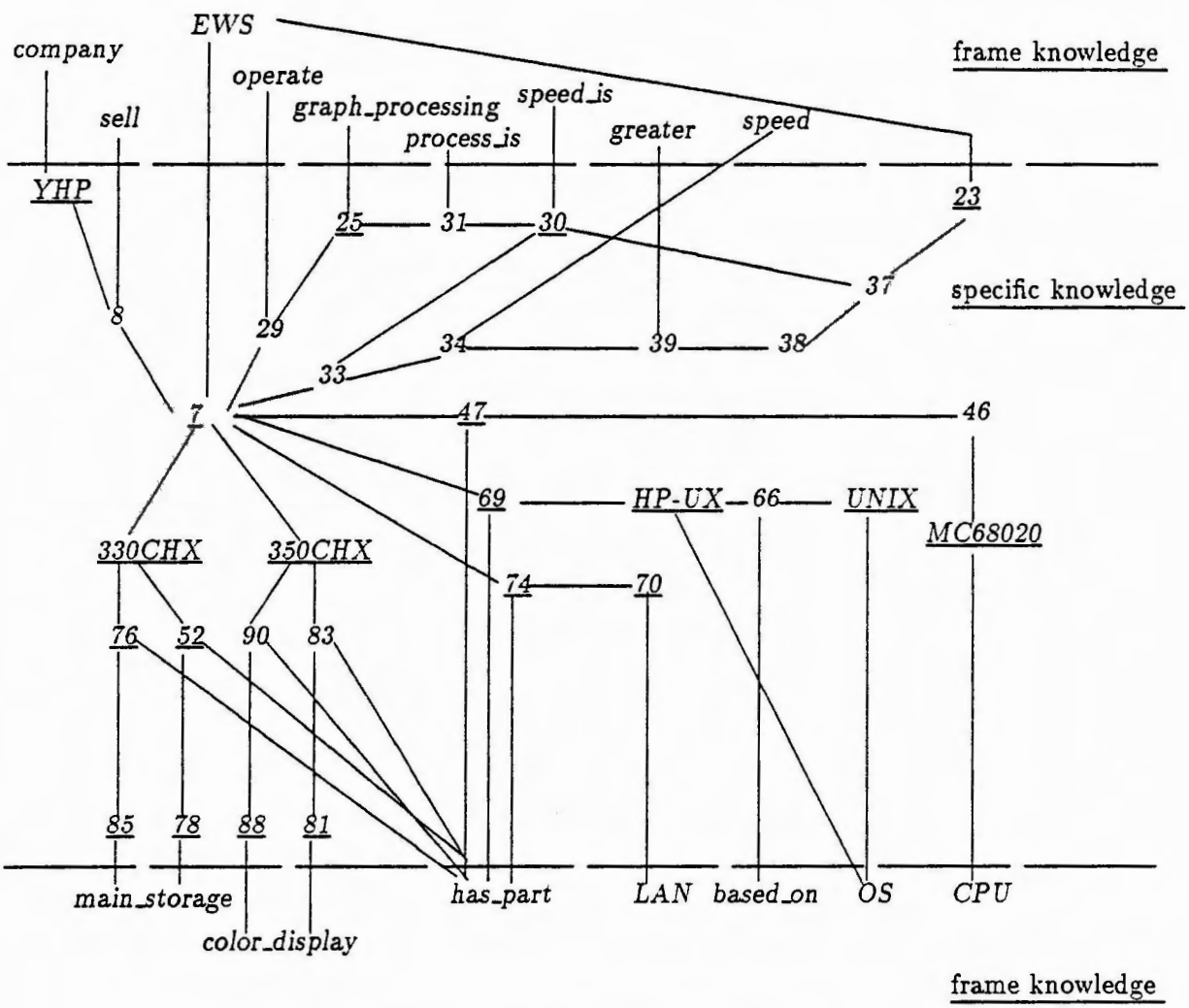


Figure 4: *representative network*

bridge of highest *importance value* . Two *islands* connected by this newly activated links are regarded as a new big *island.*

5. Repeat steps 3 and 4 until there is left one *island* only.

6. Activate each link from a node in an *island* to the nearest *frame knowledge* node. Now we can extract the *representative network* which consists only of activated links.

As an example, we applied the above algorithm to the network obtained at the example in the previous section (left hand part of Table 3). Figure 4 shows the *representative network* for the article. Underlined concepts in the figure are the top 20 concepts of higher *importance value* . Concepts without underline are nodes on *bridges*. Nodes of "name/number" type name are indicated only by the corresponding numbers because of limited space for figure. *YHP* in the figure stands for *Yokogawa_Hewlett_Packard.*

We can create the following abstract of the original article from this *representative network* , and it shows effectiveness of this method.

> *Yokogawa_Hewlett_Packard* sells two *EWSs*, *330CHX* and *350CHX*, which have *HP-UX*, a *UNIX* based *OS*, *MC68020*, and *LAN*. They also have different types of *main_storage* and *color_display*. These *EWSs* can operate some type of *graphics_processing* faster than some other *EWSs*.

6 LEARNING IMPORTANCE VALUE OF FRAME KNOWLEDGE

We have taken up a single article so far. In this section, we describe how to learn the *importance value* of *frame knowledge* while processing more than one article. The key idea is to use the *importance value* to estimate the *importance value* of another. The accuracy of *importance value* improves as the number of processed articles increases.

The biasing method described in Section 4.3 makes this incremental learning possible. That is to say, we only have to bias the constant activation values $\vec{C}$ according to the *importance value* obtained form the previous article. Just using the *importance value* for the constant activation values does not work well because Expression (2) shows that the *importance value* increases infinitely. The result is that the *importance value* of *specific knowledge* are estimated only by the distance from a concept in *specific knowledge* to some concepts in *frame knowledge* rather than by the structure of the semantic network.

To avoid this, we use the normalized *importance value* for $\vec{C}$. The normalization are done as follows by determining the maximum value of the element of $\vec{C}$ to 3.0 which is the value known from experience.

$$\begin{aligned} \vec{C}_1 &= \vec{I}, \\ \vec{C}_{k+1} &= \frac{max_c - 1}{\mathrm{MAX}_{i=1}^{n}(t_{k,i}) - 1}(\vec{T}_k - \vec{I}) + 1, \end{aligned}$$

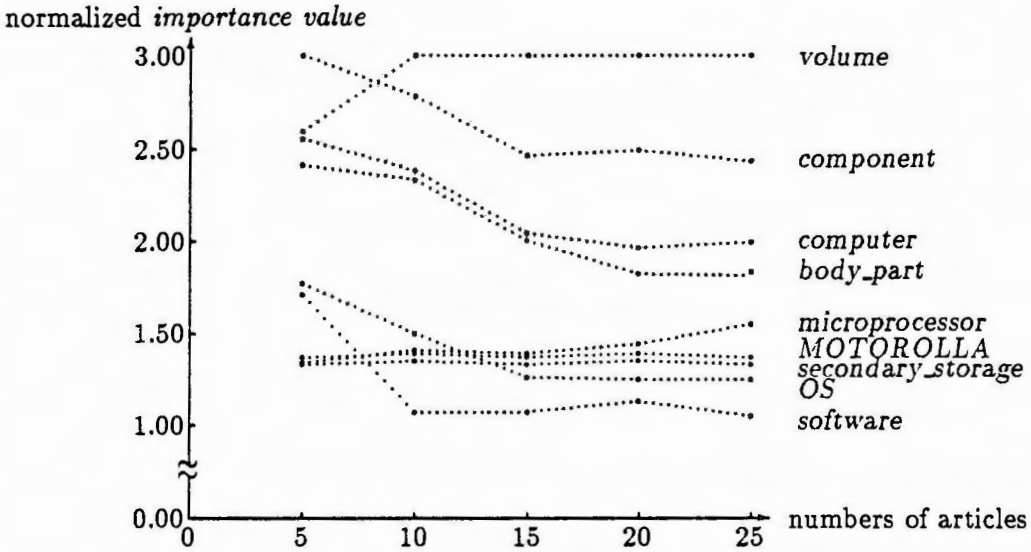


Figure 5: Changes of normalized *importance value*

where all the elements of $\vec{I}$ is 1; $\vec{C}_k$ is a constant activation value for processing the kth article; $\vec{T}_k$ is *importance value* of *frame knowledge* obtained from the kth article; $t_{k,i}$ is an ith element of $\vec{T}_k$; and *max_c* is a maximum value mentioned above.

Figure 5 shows how the normalized *importance value* of typical concepts changes as the learning proceeds. We can see the following tendency.

- The ranking of top concepts (e.g. *volume, component, computer, body_part*) get stable rather fast. It is, we think, because the main information in the articles resemble each other very much, that is, all the articles describe the *volume* of something with the *component* of *computer*.
- The concepts ranked lower (e.g. *microcomputer, OS*) show changes of ranking more clearly. For instance, the *importance value* of *OS* and *software* decrease while those of *microprocessor* and *secondary_storage* increase or be stable. This shows that the example articles lay emphasis on hardware rather than on software.
- There are concepts ranked much lower and excluded from the figure. They do not show the stable ranking because the *importance value* is so close to the constant activation values of *specific knowledge* concepts (i.e. 1) that they are influenced much more than by each article.

Learning of *importance value* s of *frame knowledge* affects how an *importance value* of *specific knowledge* is determined. We have shown the *importance value* s calculated at the initial stage of learning process in Table 2. Now we show in Table 4 how the *importance value* s of the *specific knowledge* changes if it is calculated with the learnt *importance value* s of *frame knowledge* . To

concept	before learning rank	*importance value*		after learning rank	*importance value*
EWS/isa/7	(1)	2.377	⟶	(1)	2.483
Yokogawa_Hewlett_Packard	(2)	1.598	⟶	(2)	1.598
330CHX	(3)	1.456	⟶	(3)	1.506
350CHX	(4)	1.456	⟶	(4)	1.506
graphics_processing/isa/27	(5)	1.425	⟶	(11)	1.427
speed_is/isa/32	(6)	1.391	⟶	(12)	1.392
HP-UX	(10)	1.442	⟶	(7)	1.442
UNIX	(41)	1.263	⟶	(16)	1.330
MC68020/isa/46	(42)	1.257	⟶	(15)	1.331
graphic_processor/isa/51	(43)	1.320	⟶	(19)	1.320

Table 4: Change in rank and *importance value* of concepts

clarify the difference, we list only typical concepts in both stages of before learning and after learning.

Although the ranking of *importance value* s of top few concepts are not so changed, such concepts as *graphics_processing/isa/25* and *speed_is/isa/30*, which are related to operation or speed, are ranked lower. On the other hand, such hardware related concepts as *MC68020/isa/46* and *graphic_processor/isa/51* are ranked higher according to the tendency of articles mentioned above. Such OS related concepts as *HP-UX* and *UNIX* also get higher rank because the *importance value* of *OS* is still more than 1 (i.e. the value at the initial stage) even if it is getting lower.

7 CONCLUSION

We have discussed (1) the evaluation of *importance value* by propagating signals on semantic networks with PDP techniques, (2) the representative summarization of knowledge at the concept level, (3) the incremental learning of the importance of *frame knowledge* .

This method has been proved fairly effective to measure the *importance value* when the experimental results of newspaper articles are examined. There are, however, a few problems to be solved in the future.

First, a simple linearity is assumed on signal propagation in evaluation of the *importance value* of concepts. The difference among the *importance value* for higher-rank concepts are large because of this linearity. For example, the difference between the first and second rank *importance value* is apt to become as large as shown in Figure 5. That will be modified by adopting a nonlinear signal propagation method which can evaluate *importance value* more accurately.

Second, the criteria for selecting important concepts at constructing *representative network* is set a priori at the top 20 concepts. It will be, however, possible for the system to set (or learn) the

selection criteria automatically by using the difference of the *importance value* among concepts of *frame knowledge* and *specific knowledge* .

Finally, in our experimentation, as for the *importance value* of *frame knowledge* , learning has soon saturated because the tendency of example articles resembled too much. We will make another experiment to verify the true effectiveness of this method with a larger scope of input articles. This testing would be conducted under severer condition for learning.

The learning of the *importance value* is very useful for concept learning and knowledge base managing when an attempt is made to identity the most important part among a large knowledge base. It is also supportive for generating abstract sentences in natural language processing. We would like to apply the method discussed in this paper to these fields.

References

[Handa 86] Handa, K., Higuchi, T, et al "Flexible Semantic Networks for Knowledge Representation," Journal of Information Processing, Vol.10 No.1, 1986.

[Hasida 87] Hasida, K., Isizaki, S., and Isahara, H. "A Connectionist Approach to the Generation of Abstracts," in *Natural Language Generation*, Martinus Nijhoff Publishers, 1987.

[Lebowitz 83] Lebowitz, M. "Concept Learning in a Rich Input Domain: Generalization-Based Memory," in *Machine Learning II*, Morgan Kaufmann, 1983.

[Quinlan 79] Quinlan, J. R. "Induction over Large Data Base," Technical Report HPP-79-14, Computer Science Dept., Stanford Univ., 1979.

[Sammut 83] Sammut, C. Banerji, R. B. "Learning Concept by Asking Questions," in *Machine Learning II*, Morgan Kaufmann, 1983.

Appendix A Example Article

A.1 Original Article in Japanese

横河・ヒューレット・パッカード（本社東京、社長笹岡健三氏、資本金74億円、Tel.03-331-6111）は同社の従来機種に比べて15倍の処理スピードで二次元グラフィックスをこなすエンジニアリング・ワークステーション「HP9000モデル350CHX」「同モデル330CHX」を発売した。プリント基盤やLSI（大規模集積回路）、マッピング（地図情報のコンピュータ利用）の分野向けに両機種合わせて年間700台の販売を見込んでいる。

新モデルはCPU（中央演算処理装置）に米モトローラ社の32ビットマイクロプロセッサ「68020」を採用、2.3–4.1MIPSの処理スピードを持つ。画像処理では専用のグラフィックプロセッサを搭載、縦と横だけで奥行きのない二次元ながら画像座標の変換や回転など高度な処理ができるようにした。描画速度は1秒間につき25–30万ベクタ（ベクタは描画単位）と他社機種に比べ1.5–3倍早い。

OS（基本ソフト）はAT&T（米電話電信会社）が開発したUNIXを基にした「HP-UX」を搭載、LAN（構内情報通信網）は「IEEE802・3/イーサネット」、複数画面を同時表示するウインドウ機能は「Xウインドウ」、バス（データ伝送路）は「VME」など業界標準の製品を組み込んである。

価格は標準構成のほか4メガバイトのメモリ、16インチのからディスプレーがついた330CHXが406万2千円、8メガバイトメモリ、19インチカラーディスプレーの350CHXが714万6千円。

A.2 Translated into English

Yokogawa Hewlett Packard put new engineering work stations “HP9000 model 350CHX” and “330CHX” on the market, which machines can operate 2-dimensional graphics 15 times faster than their old models. The headquarters of the company (Tel.03-331-6111) is located in Tokyo and the president is Mr. Sasaoka Kenzou. The company is capitalized at 7,400 million yen. They are expecting to sell 700 models of these machines within a year for such usage as designing printed wire or LSI and mapping (utilizing map information by computer).

The CPU of the new models are Motorolla’s microprocessors “68020”, the processing speed of which is 2.3–4.1 MIPS. The models also have a special graphic processor for translation and rotation of 2-dimensional image data, and the drawing speed therefore is 250–300 thousand vectors per second, which is 1.5–3.0 times faster than that of competitive machines.

The new models are equipped with “HP-UP”, an OS, a fundamental software, based on UNIX (developed by ATT), “IEEE8023/Ether Net” for LAN, “X-Window” for multi-window system, and “VME” bus, all of which are standard in the target area of these models.

The price is 4,062,000 yen for “330CHX” (with 4M byte memory and 16 inch color display) and 7,146,000 yen for “350CHX” (with 8M byte memory and 19 inch color display).

Appendix B IXL Statements for The Example

```
YHP = 'Yokogawa_Hewlett_Pachard',
unit(isa, NewHP, 'EWS'), unit(ako, ['350CHX','330CHX'], NewHP),
relation(sell, YHP:self, NewHP:all),
        relation(usage_is, NewHP:all, 'CAD':self),
        relation(usage_is, NewHP:all, mapping:self),
relation(headquarters_of, YHP:self, HQ:self),
        relation(located_in, HQ:self, 'Tokyo':self),
        relation(president_is, YHP:self, 'Sasaoka_Kenzo':self),
        relation(capital_is, YHP:self, 7400//'Milion yen':self),
        relation(phone_number_is, YHP:self, '03-331-6111':self),
unit(isa, Oldtype, 'EWS'), relation(sell, YHP:self, Oldtype:all),
        unit(isa, Graphic, graphics_processing),
        relation(dimension_is, Graphic:all, 2:self),
        relation(operate, NewHP:all, Graphic:all),
        unit(isa, GS, speed_is),
        relation(process_is, GS:all, Graphics),
        relation(GS, NewHP:all, X:self),
        relation(GS, Oldtype:all, Y:self),
        relation(15//times_greater, X:self, Y:self),
unit(isa, 'MC68020', 'CPU'),
        relation(develop, 'Motorolla':self, 'MC68020':all),
        relation(bit_width, 'MC68020':all, 32//bit:self),
        unit(isa, Cpu, 'MC68020'), relation(has_part, NewHP:1, Cpu:1),
        relation(processing_speed, NewHP:all, 2.3--4.1//'MIPS'),
unit(isa, graphic_processor, processor),
        unit(isa, Gp, graphic_processor),
        relation(has_part, NewHP:1, Gp:1),
        unit(ako, [translation, rotation], Graphic),
        DX = 250000--300000//'vector/sec',
        relation(drawing_speed, NewHP:all, DX:self),
        relation(competitor_of, NewHP:self, Other:self),
        relation(drawing_speed, Other:all, DY:self),
        relation(1.5--3//times_greater, DX:self, DY:self),
relation(develop, 'AT&T':self, 'UNIX':self),
        unit(ins, 'UNIX', 'OS'), unit(ins, 'HP-UX', 'OS'),
        relation(based_on, 'HP-UX':self, 'UNIX':self),
        relation(has_part, NewHP:all, 'HP-UX':self),
unit(isa, Lan, 'LAN'),
        relation(type, Lan:all, 'IEEE 802.3/Ethernet':self),
        relation(has_part, NewHP:1, Lan:1),
unit(ins, 'X-Window', window_facility),
        relation(has_part, NewHP:all, 'X-Window':self),
unit(ins, 'VME', bus), relation(has_part, NewHP:all, 'VME':self),
relation(price_is, '330CHX':all, 4.063//'Million yen'),
        unit(isa, M330, main_storage),
        relation(capacity_is, M330:all, 4//'MByte':self),
        relation(has_part, '330CHX':1, M330:1),
        unit(isa, D330, 'color_display'),
        relation(size_is, D330:all, 16//inch:self),
        relation(has_part, '330CHX':1, D330:1),
relation(price_is, '350CHX':all, 7.146//'Million yen'),
        unit(isa, M350, main_storage),
        relation(capacity_is, M350:all, 8//'MByte':self),
        relation(has_part, '350CHX':1, M350:1),
        unit(isa, D350, 'color_display'),
        relation(size_is, D350:all, 19//inch:self),
        relation(has_part, '350CHX':1, D350:1).
```

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Knowledge Acquisition Planning

Lawrence Hunter
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract:

This paper describes a technique of discovering gaps in knowledge bases through experience in using the knowledge base, and then addressing those gaps by *planning* how to find knowledge in the future experience stream that addresses those gaps. The problem can be broken up into several subproblems: explaining failures, generating knowledge acquistion goals, planning for those goals, noticing opportunities to acquire knowledge, and executing knowledge acquistion plans. IVY, a program that gains expertise in lung tumor pathology through its experience making diagnoses, demonstrates the potential of this technique.

1 Introduction

The processes of acquiring knowledge and implementing a knowledge-based system are interleaved. It is rarely possible to gather all knowledge relevant to the performance of a significant task in a single step; debugging an implemented system generally identifies gaps and inconsistancies in the underlying knowledge. Only through extensive testing and debugging of a knowledge base is it possible to bring a knowledge-based system to an expert level of performance. An important part of the knowledge acquisition process, therefore, is what might be termed knowledge augmentation or knowledge debugging.

Knowledge augmentation can be substantially automated in a case-based reasoning system. Not only can knowledge useful to the system be inferred from cases presented to the system, but the system's attention can be focused by several sources of constraint in the knowledge augmentation process that are not always available in other forms of knowledge acquisition. Since knowledge augmentation is part of a larger debugging process, information about the performance goals of the system and the details of the failure can be used to direct

augmentation. Furthermore, even a buggy knowledge base contains much information that is useful in addressing failures.

This paper describes IVY (*Interactive Videodisk Yesner*), a case-based medical reasoning system that includes an automated knowledge augmentation and debugging component. IVY is an expert consultation system for lung tumor pathology that consults by answering questions, suggesting diagnoses, and displaying images from a videodisk containing approximately 2000 pictures of lung tumors. IVY is capable of transforming its performance failures into efficient searches for new knowledge. IVY identified dozens of flaws in its initial knowledge base and, as it processed its case library, searched for needed information that it used to address many of its problems. IVY's automatic knowledge debugging process is based on powerful and flexible principles that are likely to be applicable in many domains.

2 Overview of Knowledge Acquisition Planning

IVY does knowledge augmentation by applying general knowledge acquisition principles to specific problems that arise during debugging of the system. The process of applying general principles to specific problems in knowledge acquisition is closely akin to work in planning and problem solving elsewhere in artificial intelligence (Charniak and Mcdermott, 1985) IVY's Knowledge augmentation and debugging is substantially automated by a program component that did *knowledge acquisition planning.*

Knowledge acquisition planning is similar to other forms of planning; the difference is that the plans generated specify how to notice situations which offer opportunites for gaining knowledge, how to infer relevant facts, and how to choose when and where to store acquired information. These plans are generated by mechanisms similar to those used in other forms of planning, e.g. goal decomposition or means-ends analysis. The purpose of this paper is to describe IVY's planning component and explore the potential application of work in planning to knowledge acquisition. I will first give a general overview of the process, and then describe each step in greater detail.

Planning to acquire knowledge is a three stage process in IVY. The first stage involves the detection of a performance failure, usually through expert feedback. [Failures of a somewhat different sort are also noticed when the expert's diagnoses contradicts a less expert practitioner's assessment.] When a performance failure is detected, the program explains the cause of the failure, using structures akin to Schank's explanation packages (Schank, 1986). Since IVY's domain is diagnosis, typical failure explanations specify, for example, that some entity that could be inferred from the input had been overlooked, or

that some entity had been mistakenly identified. The explanation process is implemented as a discrimination network which maps differences between the program's diagnosis and an expert supplied diagnosis to an explanation of the difference. The nodes in the network contain explanation packages whose variables are then bound to the specifics of the problem. Such explanations can be transformed directly into *knowledge acquisition goals,* or characterizations of information that would have prevented the failure. For example, if some entity had been mistakenly identified from the input, IVY would generate a knowledge acquisition goal to find a method to distinguish between the mistaken entity and the correct identification.

The second stage takes as input the knowledge acquisition goal generated by the explanation component and generates a set of plans that could be used to find the desired information. The planning process searches for general plans appropriate for the particular goal and instantiates such plans so that they address the specific content of the goal. Each plan is then checked for consistancy and executability. Plans that could not be instantiated consistantly are abandoned, and plans that are immediately executable are run. Not all knowledge acquisition plans can be executed immediately, however. Most such plans require information from specific cases of particular phenomena in order to generate the desired inferences. Plans that cannot be executed immediately are added to a pending plan list, and their applicability conditions are tested each time IVY is presented with a new case. In this manner, IVY tests each incoming case to see if that case can be used to address some pending knowledge acquisition plan.

When a plan is applicable to a particular case (or if it is applicable upon generation), it is executed. Plan execution is the third stage of IVY's learning process. Plans can specify a set of inferences to be made from a case description; a set of tests on the results of those inferences, to see if they do in fact satisfy the knowledge acquisition goal; and a set of actions to take, including modification of existing knowledge or storage of new knowledge, if the tests succeed. The execution of knowledge acquisition plans can involve making statistical inferences about features, reducing the degree of uncertainty about particular features, accumulating example cases to use as output, modifying existing inference rules to take additional information into account, or nearly any other action to modify a knowledge base. The plans organize these actions into groups, specify when that group of actions is appropriate to apply, and relate the actions to the knowledge acquisition goals that they are useful in accomplishing.

Knowledge acquisition planning is clearly a powerful tool for automating the process of debugging a knowledge base. Several questions arise regarding its practicality, however: How many and what kinds of knowledge acquisition plans are there? How domain specific are they? How expensive is the failure explanation process? What knowledge is necessary in

order to generate accurate explanations of failures? Each of these questions had to be addressed in the design and implementation of IVY. Although the generality of the approach has not yet been demonstrated, its implementation in IVY was simple and moderately successful.

3 IVY

IVY, a case-based medical expert system, has a learning component and a consultation component. In consultation mode, it parses questions from lung tumor pathologists and answers them with factual information, case descriptions and images from the videodisk library. As part of consultation, IVY is capable of describing how to recognize and distinguish among entities of medical interest, and can suggest differentials and specific diagnoses of cases presented to it. In learning mode, IVY reads structured case descriptions and attempts to diagnose them. The results of the program's diagnosis are then compared to a stored record of an expert pathologist's diagnosis. If the program's diagnosis was correct, it checks if any of the pending knowledge acquisition plans are applicable to the case, and executes any that are. If the diagnosis did not match the expert's, the program generates and plans for new knowledge acquisition goals, as described above. A simplified flowchart of IVY's learning mode appears in figure 1.

IVY's learning algorithm was tested on the cases in Dr. Raymond Yesner's library of lung pathology images. Dr. Yesner, chief of anatomic pathology at the West Haven Veteran's Administration Hospital in West Haven, CT, is one of the world's foremost lung tumor pathologists. His slide collection consists of over 2000 lung cancer related images. Dr. Jerome Silbert, Cheryl Sullivan and I produced two videodisk containing these images. In addition, the cases have been represented in machine readable form in two ways; each case has a brief description consisting of approximately a dozen descriptors, and some (about 5%) of the cases have been described in greater detail (approximately 45 descriptors per case). IVY gains its experience by diagnosing the cases in Dr. Yesner's image library.

In the course of processing the library, IVY's diagnostic and consulting performance improves significantly. Although there are few objective metrics in tumor pathology, the IVY system has received positive assessments from both both staff pathologists and experts. The improved performance level after learning was obvious to both experts and other practitioners.

IVY particularly impressed our expert pathologist by discovering a way to avoid a mistake common to inexperienced lung tumor pathologists. In doing so, IVY related two experiences that are quite different from each other in such as way as to improve its performance.

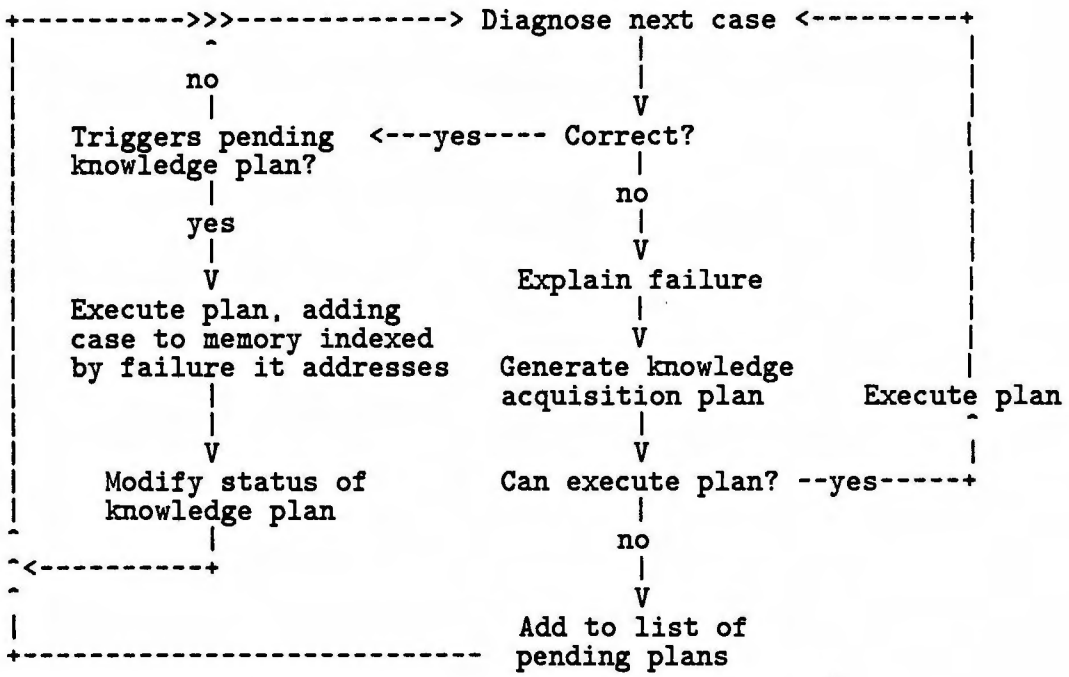


Figure 1: Flowchart of IVY's learning mode

This example demonstrates how knowledge acquisition planning differs from inductive machine learning systems and how it can lead to significant improvements in performance, so it will be described in some detail.

4 A Detailed Example

IVY's ability to plan to acquire knowledge gives it a substantial advantage over other types of machine learning. Inductive machine learning systems, the primary form of automated knowledge acquisition, compare and contrast similar experiences to form generalizations. IVY's learning method searches for useful information in any case that its plans suggest may be useful, not just in similar cases. A case that addresses a performance problem may be quite different than the one in which the problem occurred — insights often come in surprising places. One of IVY's most interesting discoveries came when it found a case that solved a problem that had arisen in a completely different context as part of a case with a completely different tumor type. This is the example case that I will use to elucidate IVY's processing. Although the example makes reference to particulars of the lung tumor domain, it is also helpful in making concrete the above, more abstract descriptions of

knowlegde acquisition planning.

4.1 The Failure

At a point midway through learning its image library, IVY failed to diagnose a particular combined squamous cell and small cell tumor, a mistake common with inexperienced pathologists. Squamous cell tumors, unlike other lung cancers, often evoke a significant immune response from the body, including large numbers of small, dark staining cells called lymphocytes in and around the tumor. It is also possible for squamous cells to be part of a combined tumor; one of those combined tumors is a mixture of squamous and small cell carcinoma. Small cell carcinoma cells are small, dark staining cells that are somewhat similar to lymphocytes. Since combined squamous cell small cell tumors account for only about 0.5% of all lung tumors and squamous cell tumors with an immune response account for more than 30% of lung tumors, it is understandable that pathologists sometimes mistake small cells for lymphocytes in that context. IVY made a similar mistake, as shown in figure 2.

4.2 Explaining the Failure

IVY generates explanations by use of explanation packages (XPs) (Schank, 1986), which are precompiled packages that capture common explanations. IVY has a library of XPs that are relevant to explaining diagosis failures. The library is segmented into four failure types, and within each failure type there are several potentially applicable XPs. The failure types are: missed entity, hallucinated entity, correct hypothesis not selected and correct hypothesis not proposed. It is simple to determine which of these failures is appropriate by comparing the program's output with the expert's diagnosis.

Within each failure type, there are a collection of XPs. Each XP has associated with it a set of preconditions under which the explanation can be assumed to hold, and a set of inferences that can be drawn if it is true. When a failure occurs, the preconditions of each XP in the appropriate failure type are matched to the failed case. If the preconditions of an XP can be matched to the failed case, the variables in the body of the XP are bound to the values of the match.

For IVY, usually only one XP in a failure type is appropriate, although there is no reason that IVY could not use mulitple explanations to generate knowledge goals (see below). However, if IVY cannot find any XP that matches, learning cannot occur from that failure. Continuing with the small cell/squamous cell example, Figure 3 shows IVY explaining the

```
IVY Learning Mode:
NEXT CASE...
PROCESSING CASE S337:
 [...]
  stroma fiberous edematous
  tissue has-parts tissue-part1 tissue-part2
  tissue-part1 tumor
  tissue-part1  cohesive whorled concentric
  tissue-cells-part1 size-large cytoplasm-amount-abundant
   cytoplasm-color-dark-pink border-distinct nuclear-shape-flattened
   irregular-nuclei
  tissue-part2 clumped loose crush-artifact nuclear-molding
  tissue-cells-part2 cytoplasm-amount-scant nuclear-chromatin-texture-fine
    nuclear-color-dark size-small
  abuts tissue-part1 tissue-part2

PROCESSING IMAGE...
Selecting features of interest... finding values...
Recognized TUMOR
Recognized CARCINOMA by NUCLEAR-CHARACTERISTICS
Recognized NON-SMALL-CELL-CA by CELL-CHARACTERISTICS
Recognized SQUAMOUS-CELL-CA by GROWTH-PATTERN CONCENTRIC and by
CONSTITUENT CELLS
 CYTOPLASM COLOR DARK-PINK
Recognized LYMPHOCYTIC-HOST-REPSONSE by CELL-CHARACTERISTICS
Recognized MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA by
 LYMPHOCYTIC-HOST-REPSONSE
No more specific kind of MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA
recognized
Checking for confusable diagnoses... none found.
IVY's diagnosis is MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA.
Dr. Yesner's diagnosis is  {COMBINED SQUAMOUS-CELL-CA SMALL-CELL-CA}.
Failure!!
```

Figure 2: IVY Misdiagnosing a combined tumor

```
Explaining failure...
Trying to account for MISSING-ENTITY SMALL-CELL-CA
Using Explanation Packages...
Finding potential explanations:
Testing explantions...
 XP:OVERLOOKED-ENTITY rejected: SMALL-CELL-CA not recognized in unanalzed
input.
 XP:CONFUSED-ENTITIES accepted: TISSUE2 might be SMALL-CELL-CA instead of
  LYMPHOCYTIC-HOST-RESPONSE.

Generating Knowledge Acquisition Goal...
 Avoid CONFUSED-ENTITIES by DISTINGUISH-BETWEEN their RECOGNTION-METHODS.

 {KG.38 Distinguish-between SMALL-CELL  LYMPHOCYTE
        in-context MICROSCOPIC MEDIUM-POWER H/E}
```

Figure 3: IVY Explaining its failure and generating a knowledge goal

failure from figure 2 by the two XPs in the missing entity failure type and accepting one of them.

4.3 Generating a Knowledge Acquisition Goal

Figure 3 also shows IVY generating a new knowledge acquistion goal. Explanations of failures are intimately related to a characterization of the information necessary to address the failure; that is why explanations are so useful. For example, if the explanation of a failure is that "X was confused for Y," then the information necessary to avoid the failure in the future is a way to "discriminate between X and Y." Or, if the explanation is "Z was overlooked in situation S," then the information necessary is a method to "recognize Z in situation S." There is an important correspondence between explanations of failures and characterizations of the information necessary to address them. IVY uses this correspondence to generate knowledge acquisition goals after it has explained a failure.

As indicated informally above, each explanation has an associated knowledge goal skeletons. When an XP is accepted for a particular failure, the variable binding in the explanation are transferred to the goal skeleton. The instantiated goal skeleton is then added to the pool of active knowledge acquisition goals.

4.4 Knowledge Acquisition Planning

IVY has a library of knowledge acquisition plans that can be used to accomplish the knowledge goals. Each goal skeleton identifies a set of plans that can satisfy it. The plans have additional applicability information stored with them, so each of the knowledge plans pointed to by a skeleton is then tested on the specific instantiation to see if the plan is applicable.

If a plan is applicable, then the roles of the plan are then filled. Each plan has a list of roles that must be filled to instantiate the plan, and a set of inference rules for finding those fillers, either from the bindings in the instantiated knowledge goal, from general information encoded in existing memory, or from the original case that caused the failure. If a potential plan is not applicable, IVY generates a message that says so, and the reason that the application failed.

Figure 4 shows the plans generated for the "distinguish between lymphocytes and small cells" goal. The DISTINGUISH-BY-FEATURE plan tries to find a feature common to both of the confused entities that has distinctive values. The program looks up the features whose values can be evaluated in the context the confusion occured and checks to see of the entites can be distinguished by any of them. In this case, the color and shape of the entities are too close to be distinguishing features, but memory does not contain exact values for their sizes. The program then generates a specific plan to find out the exact sizes of the entities, and, if the sizes are different, add size to their respective recognition conditions. The DISTINGUISH-BY-COMPARISON plan looks for an image where the two entities appear side by side. If such an image is found, it can be searched exhaustively by the program for differences, or at least can be displayed to humans who will presumably be able to find their own distinguishing features. The DISTINGUISH-BY-PARTS plan tries to find parts of the entities that are distinguishable. This plan fails in this situation because both entities have exactly the same parts.

The structure of knowledge acquisition plans is straightforward. They are divided into two parts. The first part specifies the preconditions for running the plan. These preconditions are stated in a set of terms which abstractly describe the kind of knowledge that is likely to be available. Typical knowledge acquisition plan preconditions (translated into English) are: "a case with both lymphocytes and small cells present," or "a mention of the value of the nuclear chromatin texture of an adenocarcinoma cell." Actual knowledge plan preconditions also include specifications of the type of image that the plan might apply to (e.g. acceptable magnification, stain, etc.). The second part of the plan describes the actions the plan should take. Knowledge acquistion actions involve making inferences and indexing and storing cases. English translations of typical knowledge acquisition

```
Planning for {KG.38}...
Using plans for DISTINGUISH-BETWEEN goals:

KP:DISTINGUISH-BY-FEATURE
 Checking for potential features for comparison...
 Eliminating: COLOR SHAPE
 Considering: SIZE
 Generating {KP.42}
  Preconditions: A mention in C1 of V1 <=> (value-of LYMPHOCYTE SIZE)
                 A mention in C2 of V2 <=> (value-of SMALL-CELL SIZE)
  Actions: IF v1 = v2 THEN ((DELETE-SELF))
           ELSE ((ADD-TO-RECOGNITION-CONDITIONS LYMPHOCYTE V1)
                 (ADD-TO-RECOGNITION-CONDITIONS SMALL-CELL V2)
                 (DELETE-PLANS-FOR {KG.38}))

KP:DISTINGUISH-BY-COMPARISON
 Generating {KP.49}
 Preconditions: A Case with:
                MICROSCOPIC MEDIUM-POWER H/E LYMPHOCYTE SMALL-CELL
 Actions: ((MAKE-PARADIGM-FOR (DISTINGUISH-BETWEEN
                               LYMPHOCYTIC-HOST-RESPONSE SMALL-CELL-CA))
           (INDEX-UNDER (SMALL-CELL-CA LYMPHOCYTIC-HOST-RESPONSE
                         {COMBINED SQUAMOUS-CELL-CA SMALL-CELL-CA}
                         SQUAMOUS-CELL-CA))
           (DELETE-PLANS-FOR {KG.38}))

KP:DISTINGUISH-BY-DISTINGUISHING-PARTS
 Checking for potential parts for comparison...
 Eliminating: NUCLEUS CYTOPLASM MEMBRANE
 Considering:
 No parts to consider! Abandoning plan.

Attempting to execute plans ({KP.42} {KP.48}) ... failed.
Adding plans ({KP.42} {KP.48}) to pending knowledge plans.
```

Figure 4: IVY Planning

actions are: "Store the case as a method for distinguishing small cells from lymphocytes as part of distinguishing squamous cell tumors from combined squamous cell small cell tumors," or "assume the value is typical and store it as one of the recognition conditions for adenocarcinoma cells."

4.5 An Opportunity to Execute the Plan

When the applicable knowledge acquisition plans have been generated, the plans are all tested to see if any is capable of resolving the problem immediately. In general, knowledge acquisition plans require information not available at the time of the failure, so it is uncommon for a plan to be immediately executable. Any knowledge acquisition plan that is not immediately executable is placed on a pending plan list until an appropriate opportunity arises.

In this case, no plan was immediately executable, and two plans were added to the pending plan list. At this point, IVY can do no more to address the failure, so it moves on to the next case. In the processing of all future cases, IVY checks the applicability conditions of each pending plan to see if there might be an opportunity to execute it. In this way, experiences in the future can be tapped specifically to learn to avoid past problems.

IVY treats every incoming case as a potential opportunity to satisfy one of the pending plans. Each incoming case is matched against the preconditions of all potentially relevant pending knowledge acquisition plans. When a case matches the preconditions of a plan, that plan is executed. It is important to note that the case that triggers the execution of a knowledge acquisition plan is not necessarily similar to the case that generated the original failure. The case that triggered execution of the plan embodies the information necessary to *avoid* such a failure. That information may become available in many possible cases. Once a case that addresses the problem (i.e. allows a plan to be successfully executed) is found, that case can be linked to the case that embodied the problem. Now when a potential failure similar to the original case is encountered the program can respond with useful information.

IVY processed dozens more cases that were not relevant to the combined tumor versus host response problem before coming across a case that met the applicability conditions of one of the knowledge acquisition plans for it. The processing of that case is shown in figure 5 on the previous page.

The case that triggered the plan is a metastasis of a small cell carcinoma to a lymph node. There are many lymphocytes in lymph nodes, so many are visible in the image. The juxtaposition of lymphocytes and small cell tumor cells in a single image makes their

```
NEXT CASE...
PROCESSING CASE S1381 ...
  [...]
IMAGE-TYPE M
IMAGE-SOURCE auxillary lymph-node
IMAGE-PROCEDURE biopsy
  [...]
 tumor loose nuclear-molding
 tumor-cells size-small cytoplasm-amount-scant nuclear-color-dark
   nuclear-shape-oval nuclear-chromatin-texture-fine
 lymphocytes amount-moderate texture-clumped
 admixed tumor-cells lymphocytes

PROCESSING IMAGE...
Selecting features of interest... finding values...
Recognized TUMOR
Recognized CARCINOMA by NUCLEAR-CHARACTERISTICS
Recognized SMALL-CELL-CA by CELL-CHARACTERISTICS and GROWTH-PATTERN
Recognized WELL-DIFFERENTIATED SMALL-CELL-CA by NUCLEAR-CHARACTERISTICS
No more specific kind of WELL-DIFFERENTIATED SMALL-CELL-CA recognized
Checking for confusable diagnoses... none found.
IVY's diagnosis is  WELL-DIFFERENTIATED SMALL-CELL-CA
Dr. Yesner's diagnosis is  WELL-DIFFERENTIATED SMALL-CELL-CA
OK.
Checking pending knowledge plans...
 {KP.49} preconditions met!
  This case has MICROSCOPIC MEDIUM-POWER H/E LYMPHOCYTE SMALL-CELL
  Executing knowledge plan...
   Labelling case S1381 as a paradigm for (DISTINGUISH-BETWEEN
    LYMPHOCYTIC-HOST-RESPONSE SMALL-CELL-CA)
   Storing S1381 under:
     A method for avoiding the problem in case S337: CONFUSED-FOR
       SQUAMOUS-CELL-CA
     Also storing S1381 under:
       An example of SMALL-CELL-CA
       An example of LYMPHOCYTIC-HOST-RESPONSE
       A method for resolving DIFFICULT-CASE of SMALL-CELL-CA
       A method for avoiding CONFUSED-FOR LYMPHOCYTIC-HOST-RESPONSE
       A method for resolving DIFFICULT-CASE of
         {COMBINED SQUAMOUS-CELL-CA SMALL-CELL-CA}
   Deleting other plans for {KG.38 Distinguish-between SMALL-CELL
      LYMPHOCYTE in-context MICROSCOPIC MEDIUM-POWER H/E}
Tested all pending plans...
```

Figure 5: IVY finds an opportunity and executes its plan

differences apparent to a pathologist (although, in this case, not to the program itself). Even though the program cannot infer a clear set of distinguishing features, it can infer that this image is helpful to a pathologist who needs to make the distinction. The knowledge acquisition plan specified that this case be stored not only as an example of a small cell carcinoma, but also as a case is relevant for avoiding a confusing a squamous cell tumor for a combined squamous cell small cell cancer.

4.6 A More Expert IVY

IVY found a case it could use for differentiating between small cells and lymphocytes. The plan that identified the case also specified how to store it so that it IVY could use it later, as in figure 6, shown on the next page.

When IVY later encountered another squamous cell tumor with a separate population of small dark cells, the recognition conditions for pure squamous cell carcinoma were again triggered, but this time there is an entry in "confused-for" list associated with the diagnosis. This is a list of difficult diagnoses and cues that they might be relevant. The small darks cells that could be small cell carcinoma cells were placed on this list when a distinction method was found.

Because the input case matched an entry on the confused-for list, IVY generated a warning that the input case could be either a squamous cell tumor with a lymphocytic host response or a combined squamous cell small cell tumor. It then offered to display a typical squamous cell tumor, a typical combined tumor and the small cell and lympocyte images to help make the distinction. Although IVY could suggest only several possible diagnoses instead of making the correct one, it did not repeat its error, and it did present appropriate and useful information that it had learned.

The the case for distinguishing small cells and lymphocytes also turned out to be useful in another circumstance. In diagnosing another case, IVY encountered a set of small, dark cells with scant cytoplasm in an adrenal gland. The cells met the recognition conditions for both lymphocytic host response and small cell carcinoma. IVY suggested both possible diagnoses and displayed the paradigm distinction case. By storing the paradigm case in the appropriate manner, IVY improved its performance both in cases like the original failure and in situations that were significantly different than the original. Once an experience has been characterized as useful for a particular task, it becomes possible for a program to apply that experience in a wide variety of situations that would be hard to recognize on the basis of similarity alone.

```
IVY Consultation Mode:
Ivy> Diagnosis
DESCRIBE THE IMAGE, VIEW FIRST, USING STANDARD IVY TERMS.  2 <CR>S TO END.
Ivy.dx> microscopic medium-magnification h-e lung biopsy
Ivy.dx> tissue has-parts tissue-part1 tissue-part2
Ivy.dx> tissue-part1 tumor
Ivy.dx> tissue-part1  cohesive concentric desmosomes
Ivy.dx> tissue-part1-cells size-large cytoplasm-amount-abundant
cytoplasm-color-dark-pink
          irregular-nuclei
Ivy.dx> tissue-part2 clumped loose
Ivy.dx> tissue-part2-cells cytoplasm-amount-scant size-small nc-ratio-high
nuclear-color-dark
Ivy.dx> abuts tissue-part1 tissue-part2
Ivy.dx>

PROCESSING IMAGE...
Selecting features of interest... finding values...
Recognized TUMOR
Recognized CARCINOMA by NUCLEAR-CHARACTERISTICS
Recognized NON-SMALL-CELL-CA by CELL-CHARACTERISTICS
Recognized SQUAMOUS-CELL-CA by GROWTH-PATTERN DESMOSOMES and by CONSTITUENT CELLS
 CYTOPLASM COLOR DARK-PINK
Recognized MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA by GROWTH-PATTERN
 DESMOSOMES
Recognized LYMPHOCYTIC-HOST-REPSONSE by CELL-CHARACTERISTICS
No more specific kind of MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA
recognized
Checking for confusable diagnoses...
 SQUAMOUS-CELL-CA with ABUTS LYMPHOCYTIC-HOST-RESPONSE has been confused for
   {COMBINED SQUAMOUS-CELL-CA SMALL-CELL-CA}!
Looking for distinction method... found!

Possible diagnoses are:
 MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA
 {COMBINED SQUAMOUS-CELL-CA SMALL-CELL-CA}

Distinguish between the diagnoses by distinction between SMALL-CELL-CA and
 LYMPHOCYTIC-HOST-RESPONSE.  Compare the following images:
Image 1 is a BIOPSY of LUNG MEDIUM-MAGNIFICATION H/E view of a
 MODERATELY-WELL-DIFFERENTIATED SQUAMOUS-CELL-CA for recognition of
 SQUAMOUS-CELL-CA.
Image 2 is a BIOPSY of LUNG MEDIUM MAGNIFICATION H/E view of a
 {COMBINED SQUAMOUS-CELL-CA SMALL-CELL-CA} for recognition of
 COMBINED-CA
IMAGE 3 is a BIOPSY of AUXILLARY-LYMPH-NODE MEDIUM MAGNIFICATION H/E view of
 SMALL-CELL-CA for distinction between SMALL-CELL-CA and
LYMPHOCYTIC-HOST-RESPONSE
To see any of the images again, type the number of the image.  Type <CR> to
continue.
```

Figure 6: IVY after learning

5 General Considerations in Knowledge Acquisition Planning

IVY's use of knowledge acquisition planning to augment its knowledge base offers several general lessons. First, IVY demonstrates both the possibility and utility of planning to acquire knowledge. IVY began with (hand coded) baseline performance on par with a textbook-trained young pathologist, and eventually developed a well indexed library of experiences that enabled it to impress an expert pathologist with its performance. The experiences it selected for storage and the indexing method it used both depended entirely on the knowlede acquisition planner.

Second, IVY shows that machine generated explanations of performance failures are an appropriate source of knowledge acquisition goals. Explanation-based learning can be used to identify sources of error in existing knowledge bases. Once identified, such sources of error can be used to focus a search for additional information that would address the error. This combination of explanation-based learning and automated knowledge acquisition can make a contribution to both fields. (See Hunter, 1988 for a description of the relevance of IVY to explanation-based learning research.)

Third, IVY provides evidence that knowledge acquisition plans may be somewhat domain independent, and that it may be possible to create a generally useful knowledge planning library. Some of IVY's plans can be paraphrased in terms like "To distinguish between X and Y, find a case that contains clearly visible representations of both X and Y and compare them side by side," or "To find a more accurate method to recognize X, find a feature of X whose expected value is not specified or is not specified as concretely as possible, and use a sample of cases to specify its expected value." Such very general plans must be specialized for particular domains so that they are applied appropriately, but the relatively small degree of specialization required for IVY suggests that there may be an important connection between domain independent learning methods and knowledge based learning methods.

One important feature that differentiates IVY from other machine learning systems is what information the program is looking for and the methods used in the search are under program control. In this way it becomes possible to dynamically tailor general learning methods to the specific needs and resources of the learner. Progams that can change what they are trying to learn and the methods they are using to learn it have substantially greater flexibility than those in which the "target concepts" to be learned are fixed by the implementor. Learning programs with dynamic attention focus hold substantial promise for automating knowledge augmentation.

Finally, it is worth considering the analogy between this system and the way people learn. Knowledge acquisition planning requires both general planning knowledge, some specific domain knowledge and a set of experiences with expert feedback. The result is that gaps in the domain knowledge that cause performance problems are detected by the system, and, with appropriate experiences, can be remedied automatically. This bears some resemblence to the manner in which human professionals are typically trained: Students with suitable general knowledge are taught the fundamental domain knowledge in courses and then are given supervised experience where they can detect problems in their understanding and extend their knowledge of the domain through experience. Although IVY is not intended as a detailed psychological model, its construction was strongly influenced by considerations of psychological plausibility,

6 Future Directions

IVY is just a first step in developing more sophisticated knowledge acquisition planning systems. The purpose of this section is to propose several potential modifications that may make this work more generally applicable or increase the power and efficiency of such planners. Such suggestions are, of course, inherently speculative.

One extension that seems immediately desirable would be to extend IVY's planner to be able to take into account expected costs and benefits in making its choice about which plans to instantiate. Such an extension could be integrated into the existing IVY framework because IVY already makes decisions about what methods to use for which goals; the extension would add additional considerations to that decision, but would leave the structure of the system intact. This modification might be a significantly advance toward one of IVY's underlying design goals, namely providing a mechanism for appropriate focusing of attention during learning.

Another simple step toward improving IVY's ability to focus its attention appropriately would be a system for restricting which pending plans are tested against a given input. A modified RETE algorithm (Forgy, 1979) might be usefully applied to the preconditions of knowledge plans, but even stronger constraints might be applicable. It may be useful to link the restricitveness of knowledge plans' preconditions to their importance, or to their cost/benefit ratios.

One of IVY's most significant flaws is that it never considers the interactions of its plans or goals. Pending knowledge plans can accumulate very rapidly, and checking each plan against every incoming case can become quite expensive. Furthermore, human learners often notice patterns in their mistakes and then look for information that might address

some broader or more systematic error. Perhaps it would be possible for a system like IVY to attempt to integrate new plans into existing ones before posting them to the pending list. Planners in other domains have embodied systems for combining plans that have similar preconditions; knowledge acquisition planners might be more concerned about combining plans for similar goals.

Other research on planning might also be useful in for this task. For example, Birnbaum, 1986 contains many important results relevant opportunistic planning. Knowledge acquisition plans could be written to retrieve information from databases or other on-line reference sources. A program that could use such plans could find desired information from a wide variety of sources on demand.

7 Conclusion

IVY is only the first attempt at exploiting the extremely promising combination of automated knowledge acquisition with automated planning. The AI literature has touted general planning as the solution to many problems. Whenever limited resources need be applied to accomplish particular ends, planning is appropriate. Considering knowledge acquisition as the application of limited inferential resources to ends dictated by performance needs it becomes clear that general planning strategies may offer significant advances. The task now is to generate planning techniques specific to acquiring knowledge, and exploring the pragmatic difficulties with implementing such schemes.

8 References

Birnbaum, L. A. (1986) *Integrated Processing in Planning and Understanding*, PhD. thesis, Yale University.

Charniak, E. and McDermott, D. (1985) *An Introduction to Artificial Intelligence*; Reading, MA: Addison-Wesley.

Forgy, C. L. (1979) *On the Efficient Implementation of Production Systems*, Technical Report, Carnegie-Mellon University, Computer Science Department.

Hunter, L. (1986) Explanation-based Discovery, in the *Proceedings of the AAAI Symposium on Explanation-based Learning*, American Association for Artificial Intelligence, Stanford, CA.

Schank, R. (1986) *Explanation Patterns: Understanding Mechanistically and Creatively*; Englewood Cliffs, NJ: Lawrence Erlbaum Associates.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

DRAFT

KNOWLEDGE REPRESENTATION FOR KNOWLEDGE ELICITATION

(ABSTRACT)

AUTHORS: N.E. JOHNSON C.M.TOMLINSON

Department of Computer Science
Brunel University
Uxbridge,
Middx UB8 3PH
U.K.

E-Mail JANET N.Johnson@cs.Brunel.ac.uk

0. INTRODUCTION

This paper addresses both practical and theoretical needs of knowledge elicitation/acquisition. The first section looks at the needs of the full custom knowledge engineer who, in an unfamiliar domain, and aware that a first generation expert system shell will not be adequate to the task, begins the stage of conceptualizing the domain. Even with a clear committment to a knowledge engineering environment neither the knowledge engineer nor the expert will be willing to allow the strong demands of implementation languages to shape the knowledge early in the elicitation process. Yet some form of representation beyond pencil and paper notes is required. It is this problem of trying to document 'emergent' knowledge which is a genuine practical difficulty for any knowledge engineer at the beginning of a project and for which there is as yet very little in the way of computer support.

The second section takes up the theoretical problem of what makes for good practice in knowledge elicitation. Here an argument is given for taking a more rigorous approach to the documentation of 'emergent' and 'mature' knowledge which involves considering the need to validate or otherwise test newly elicited knowledge. This approach draws on the notion of treating knowledge elicitation as something akin to a research activity of analysing the qualitative data which is the product of using a primary knowledge elicitation technique (e.g interviews, problem solving observation studies etc.) It is suggested that the representation form should mediate between this qualitative data and the implementation language for the KBS. The idea of a 'mediating representation', distinct from an implementation led representation (a knowledge representation language), is explored in terms of some basic requirements of knowledge elicitation (expressiveness, ease of communication etc). This approach draws little from the AI or Cognitive Science approach to the modelling of human expertis. We are seeking to give adequate descriptions of expert's tasks or expertise which rely on grammar-type rather than cognitive models.

The third and fourth sections contains a report on current work on using a mediating representation based on Systemic Grammar Networks (Halliday et al **) to represent knowledge. First the formalism is described. Then a toy example given to demonstrate how it may be used in practice. Subsequently a real example is described from current work in the european project ESPRIT TAO (a Therapy Advisor in Oncology).

Finally, in the postscript some remarks are made on how some consideration of the concepts explored above may be useful in designing future knowledge based systems for synthetic problems.

(This abstract contains extracts from previously published work.)

1. THE NATURE OF KNOWLEDGE ELICITATION

Two arguments are given here, the first from general observations on the 'creative' nature of knowledge elicitation and the second from the overlap of knowledge engineering and software engineering with respect to requirements analysis.

Elicitation involves creating an environment where an expert, and others, can

generate some kind of description of their activities which the knowledge engineer comments upon, analyses and moulds into a body of 'knowledge'. Thus knowledge elicitation is not the discovery of heaps of mature, internalised cognitive structures; nor is it the mapping of incompletely expressed concepts into the seductive precision of a formal system. It is closer to a learning or research activity where one, usually a knowledge engineer, comes to understand something of the concerns of the other (the expert). With varying degrees of appropriateness, the knowledge engineer actively creates the knowledge from a sea of qualitative data produced during their meetings. The two stages of elicitation, raw elicitation (interview, problem solving etc) and knowledge analysis (getting it down on paper) are distinct but do co-occur and knowledge engineers need skills in both. Skill in using Interview techniques is widely acknowledged to be valuable. More rarely is it admitted that skill in the use of Mediating Representations is an essential part of the knowledge analysis stage. For a description of common pencil and paper methods see Martin & McClure (1986) and SDL INSIGHT Study 6. For a more complete discussion of the notion of a Mediating Representation see Johnson, NE 1987b).

The experienced developer of first generation expert systems will have become adept at 'rule spotting' and feels no need of any extra representation device beyond conceptual hierarchies and decision trees. In this situation the rule base is likely to be the only substantial version of the knowledge to survive the project. So, only when the system is to be enhanced, ported to another machine or the original rule base writer moves on, will the KBS team wish they had a machine independent form of the knowledge to reimplement the system. Redoing the knowledge elicitation is a known, if desperate, solution to a lack of adequate knowledge documentation!

Even more difficult is the problem of the full custom knowledge engineer, working in a new domain and aware that a first generation architecture, typically an expert system shell, may not be adequate. What is needed is a facility for capturing what may be relevant and preserving it until a selection of implementable knowledge can be made. Using standard knowledge representation languages (Production Systems, Frame based languages, Associative Semantic Networks, Lisp like structures etc) is already to prejudge and thereby eliminate features of the expertise or knowledge which, with hindsight, ought to have been implemented.

There is thus a clear parallel here with good software engineering practice in that care taken during requirements analysis (= elicitation) will pay dividends in time and effort during the later stages of prototype implementation. Unfortunately knowledge engineers cannot look to software engineering for an off the shelf solution to the elicitation problem, for the stage of requirements analysis is the least well understood stage of software design and thus the least well supported area. While the technology is available it is clear we have still to develop our own methods.

In section 2 it will be argued that the available automated knowledge acquisition tools, including sophisticated knowledge base editors, do indeed support the 'knowledge book-keeping' tasks but do not support the conceptually difficult tasks of early conceptualization of the domain nor the later stage of validating the knowledge. Both are essential. The first because early elicitation is 'creative' and so can easily degenerate into an undisciplined selection of 'knowledge' depending on that which the knowledge engineer understands or that which the system can accept. The second is

important because good practice in KBS and expert systems development demands that claims made for the validity of the knowledge in the system be tested against a wider population than the original expert. To this end we suggest the use of a structure like a Systemic Grammar Network (SGN see section 3) as a good tool to help proceed in a disciplined yet flexible manner.

2. GOOD PRACTICE IN KNOWLEDGE ELICITATION - KNOWLEDGE STRUCTURING AND MEDIATING REPRESENTATIONS

THE COMPUTATIONAL MODEL METAPHOR AND THE GRAMMATICAL MODEL METAPHOR

The Computational Model is the default model in Knowledge Engineering. In this section we summarise the effects on knowledge elicitation through illustrations from the use of current methods and systems. Examples will includethe use of machine oriented formalisms, (IF-THEN rules, Associative Semantic networks, AI style of human expertise) and as poor candidates for documentation of emergent knowledge. Then the Grammar Metaphor will be advanced as an alternative, and more appropriate modelling technique in knowledge elicitation, with outlines of requirements on knowledge elicitation tools in general (eg expressive formalisms, limitations on power of grammar...). One requirement is the Mediating Representation.

MEDIATING REPRESENTATIONS

What They Are!
A Mediating Representation for knowledge elicitation is a computer language independent notation used as a conceptual aid in synthesising knowledge from talk with experts. It 'mediates' between verbal data and standardised knowledge representation schemes found in AI software development environments. For the group at Brunel the major Mediating Representation is the Systemic Grammar Network (Halliday 1978, Winograd 1983 and Bliss, Monk & Ogborn 1983). This formalism is a key feature of the process of abstracting and conceptualizing expertise in a domain. It provides us with the means to prepare an explicit description of the knowledge in a definitive form for discussion within the KBS development team. It also allows us to attempt validation of would-be-knowledge as part of a more rigorous approach to knowledge elicitation (Johnson & Johnson 1987).

What They Are Not!
Firstly, a Mediating Representation is not a 'paper knowledge base'. The paper knowledge base is a more restricted form of representation of some of the data from elicitation. It is a computer based representation scheme, perhaps with loose collections of 'other relevant information'. A paper knowledge base is a focus of activity during elicitation but the aim is to provide a communicable representation of the knowledge base only. A Mediating Representation is the major tool in constructing a machine-independent statement of the knowledge and associated background information called a Competence Model (Keravnou & Johnson 1986). The aim is to provide both a medium of communication between members of the team and a 'grammar' of the expert's task.

Secondly, a Mediating Representation is more than a version of the linguists', and latterly knowledge engineers', notion of 'intermediate

representation'. An intermediate representation has a more temporary status. It is a transient, indeed intermediate, repository for the knowledge which becomes redundant once the knowledge has found expression in either another intermediate representation or in the final form of codable knowledge. We ask our mediating representation to be more in that it should hold a permanent record of the structure of the knowledge (the grammar) from which several selections made be made of implementable knowledge. It acts as a communication medium between all members of the knowledge engineering team at all stages of system building and evaluation. In addition we prefer the term Mediating Representation to convey the sense of synthesis and coming to understand through the representation which is constitutive of knowledge analysis.

WHAT IS EXPECTED OF A MEDIATING REPRESENTATION?

Explicitness and Expressiveness

Terms which express formal features of a representation scheme vary in how well they reflect or make transparent features of the data in question. A pre-selected collection of terms (the vocabulary of set theory, the family of terms relating to actors and agency or 'is-connected-to' type links) can be well defined but may not be appropriate to express the detail or colour of expertise. Special purpose terms made up of slots or well chosen names will be more expressive but of course can be simply idiosyncratic and therefore not explicit. For example, the basic If-Then format of rules in expert systems shells is often too bland to reflect explicitly all the shades of knowledge we may want to implement. (See compiled knowledge in MYCIN Rules e.g. Keravnou & Johnson 1985.) The richer source of terms in Associative Semantic Networks looks more promising but such networks can have as many types of nodes as types of links and so resemble conceptual spaghetti (a generic example will be given).

Economy and Power

This is a notion borrowed from linguistics that says a grammar should be economical in the sense that it assigns structure to all and only the sentences of a language. Thus our mediating representation must capture all the relevant facts, opinions and relationships in the domain and exclude nonsense and contradiction. (Relevant here means with respect to the expert's conception). Representation schemes which generate conceptual absurdities or errors of fact are generally too powerful for the data. Thus an induction algorithm, which even on a restricted set of examples still generates unsympathetic rules, is not providing a good grammar (underlying structure) of the knowledge.

Communicability

The major distinguishing feature of a Mediating Representation scheme and a Knowledge Representation language is the purpose to which each is put. A Knowledge Representation language is a machine oriented scheme where the demands of implementation are entirely appropriate. The purpose of a Mediating Representation is to provide a 'grammar' of the knowledge for communication between humans in order that appropriate choices of architecture, knowledge representation language and MMI features may be made. It is suggested here that a graphical format is necessary, although never sufficient. Hence it comes as no surprise to hear that experts find critiquing individual rules or other forms of pseudo code to be an uncongenial experience. Decision trees seem to fare better, but only when pruned! It is

now even possible to display knowledge bases in the form of semantic networks on screen with integrity preserved under change. Unfortunately, as we argued in section , the facility to draw and redraw complex diagrams on screen does not guarantee that well structured accounts of knowledge will emerge. A picture of knowledge, no matter how detailed, cannot be a grammar. Nor will it necessarily make knowledge transparent to anyone else on the KBS team. Thus indiscriminate use of sophisticated graphics editors will not make knowledge elicitation, as a conceptual task, very much easier.

Guidance in Knowledge Analysis

In section 2 it will have been suggested that a sophisticated graphics editor, although desirable, can lead the unwary into producing visually and conceptually confused 'pictures' of knowledge. For the notion of Mediating Representation advanced in this paper we too require the 'pictures' to be visual displays. But they are to be displays of grammar like structures which reveal important features of the qualitative data arising from elicitation not just human oriented displays of machine readable code. For the sake of good practice in knowledge elicitation we require the facility to be able to test the structure against the data for fit and descriptive adequacy. In the real example of section 4 we show how this was done in practice.

In principle, we require the Mediating Representation to do two things:

support the process of deriving data driven concepts

generate a description of the knowledge which can be tested against the original data for completeness and soundness (and subsequently against external sources).

In the first requirement, the Brunel group have adopted an approach based on Grounded Theory (Glaser & Strausss 1967) whereby transcripts of interviews are read closely with much indexing, collating and memo writing to generate simple abstract terms which lift the analysis one step from the transcripts but with the analysis still 'grounded' in the original data. This is one way of tackling the gritty problem known to social scientists as 'forming data driven concepts' or 'qualitative data analysis'. In these circles the problem is perceived to be one of analysis at a conceptual or semantic level. In knowledge engineering the problem has often been approached through Protocol Analysis ot 'transcript analysis' systems (e.g. KEATS) which only offer protection to syntactic analysis. These approaches would best be regarded as tools to clean up after a conceptual analysis. It can be argued that before entering any of the current KBS development environments there is always a conceptual analysis stage which is as yet unsupported and for which a Mediating Representation is essential. For it seems that most knowledge engineers make fundamental choices about the character of the expertise well before entering the KBS environment, usually driven not by the data, but by what can be implemented in that particular environment. (cf early work with KADS).

Using the approach of Grounded Theory to generate names for distinctions in the domain and expertise and encapsulate those terms in a Systemic Grammar Network (SGN). The SGN can be traversed to generate'descriptions' of events or circumstances which ought to represent genuine points of interest in the domain. For example, an SGN which characterises strategies of chip designers could generate descriptions of possible histories of design decisons. These possible histories are suggested by and checked against the original interview transcripts and subsequently shown to other chip designers to 'test' for completeness and soundness. These networks display, in a concise

graphical form, nodes whose names reflect (expressively!) the salient features of the domain and expertise. The few and simple links have the advantage of making the displays easy to absorb. The pseudocode generated by traversing the SGN or picking out combinations of terminal items should describe all and only possible events or 'objects' in the experts task. Thus our SGN's with supporting glossaries are custom built to provide a grammar of the task. By accepting the discipline of producing a grammar it is possible to be guided in a principled fashion through the knowledge analysis phase. By repeatedly asking the questions

Does our expert approve of the choice of terms used in the SGN?

Does the total set of combinations (the paradigms) adequately represent the possibilities the expert described during elicitation?

Do the sequences of pseudocode generated by passing through the network generate sensible characterisations of real events or objects with respect to the domain?

Do the 'failures' say something about the SGN or about holes in the elicitation data?

a series of SGNs will evolve which capture much more of the flavour of the expertise and domain than is normally believed necessary. The short argument is that if explicitness is desirable, then a Mediating Representation is an essential tool and it should both guide the process and provide a fertile structural format.

4. SGNs - AN EXAMPLE OF A MEDIATING REPRESENTATION

A Systemic Grammar Network is a well defined system, based on the idea of choice, with no standard rules of application. Essentially the notation provides an empty structure into which a chosen domain of knowledge may be fitted. The 'content free' of networks is their most useful feature in knowledge analysis. Like other networks, a systemic network is a graph composed of nodes joined by links. All nodes are names invented by the analyst but suggested by the data. The most commonly used links are of two types, a mutually exclusive choice, denoted by a straight bracket and a co-occuring choice, denoted by a curly bracket as in Figure 2. (There is also a recursion symbol and a conditional link.)
For the purpose of illustrating the heuristic use of SGNs we will give a simple example on a domain appropriate to a US audience.

4. A REAL EXAMPLE

This section will contain an example of an SGN from our current research in ESPRIT 1592 TAO (A Therapy Advisor for Oncology).
The example will be about how to decide whether a patient can receive a specific course of chemotherapy (inclusion in the protocol).
Comments will be makde on how this SGN was used

as a knowledge elicitation tool

as a medium of communication with the knowledge implementation team

how the SGN relates to the system architecture in terms of task trees, intelligent data handling and strategic explanations

plans for testing the validity of the knowledge with other experts.

5.POSTSCRIPT

The above is of real concern to full custom knowledge engineers. It is not so obvious to the applications development team, working with a familiar domain and familar tools, that so much effort should be put into documenting knowledge. To that audience we would say that our techniques are overblown for their current needs. We are not suggesting that the mediating representation is the 'silver bullet' of knowledge elicitation. But we do doubt the claims made for automated knowledge acquisition systems to handle the next generation of synthesis problems (design etc). The need is not for yet more sophisticated technology, for what we already have is under employed. What we lack are sophisticated methods. In short our concern is with tools and methods which will allow knowledge engineers to scale up to more complex and complete KBS.

BIBLIOGRAPHY

Banff Conference 86
Banff Conference 87

Bliss,J. Monk, M. & Ogborn,J. (1983) Qualitative Data Analysis: a guide to the use of Systemoic Networks, Croom Helm, London.

Chandrasekaran, B. (1986) Generic Tasks in Knowledge Based Reasoning. IEEE Expert, 1,3 pp23-32.

Clancey, W.J. (1984) Acquiring, Representing & Evaluating a Competence Model of Diagnostic Strategy. HPP Memo 84.2, Stanford University. To appear in Contributions to the Nature of Expertise, Chi, M. Glaser, R. and Farr, M. (eds).

Glaser, B.G. & Strauss, A.L. (1967) The Discovery of Grounded Theory: Strategies for Qualitative Research. Chicago, Aldine.

Halliday, M.A.K. (1978) Language as a Social Semiotic. Edward Arnold, London.

Johnson, L. & Johnson N.E. (1987) A Knowledge Elicitation Method for Expert Systems Design. Int J. Systems Research and Info Science. Vol. 2. Gordon & Breach, London.

Johnson, N.E. (1985) Varieties of Representation in Eliciting and Representing Knowledge for IKBS. Int. J. Systems Research and Info. Science 1,2 pp69-90.

Johnson, N.E. (1987a) Knowledge Elicitation In Practice, INSIGHT STUDY NO.6, Systems Designers Scientific, Pembroke Broadway, Camberly, Surrey GU15 3XD, U.K.

Johnson, N. (1987b) Mediating Representations in Knowledge Elicitation. EKAW87, Reading University, U.K.

Johnson, N. (1988a) Knowledge Representations in Knowledge Elicitation. to appear in Diaper, D. The pop up book of Knowledge Elicitation (-or something like that!)

Johnson, N. Tomlinson CMT. (1988b) Knowledge Elicitation for Second Generation Expert Systems. EKAW 88 (Bonn).

Keravnou, E.T. & Johnson, L. (1986) Competent Expert Systems: a case study in fault diagnostics. Kogan Page, London.

Martin, J. & McClure, C. (1985) Diagramming Techniques for Analysts and Programmers. Prentice Hall.

Winograd, T. (1983) language as a Cognitive Process. Addison Wesley, London.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

PROGRESS IN AUTOMATED KNOWLEDGE ACQUISITION TOOLS: HOW CLOSE ARE WE TO REPLACING THE KNOWLEDGE ENGINEER?

Catherine M. Kitto
Knowledge Systems Laboratory
Advanced Technology Center
Boeing Computer Services
P.O. Box 24346, M/S 7L-64
Seattle, Wa. 98124
email: cathy@boeing.com

ABSTRACT

Knowledge acquisition tools have been developed to provide automated assistance in the construction of knowledge-based systems for problem-solving. They elicit and model knowledge from a human domain expert and automatically generate the knowledge-based system. One of the important goals of many of these tools is to reduce or eliminate the role of the knowledge engineer in the development of these expert systems We will examine the issue of the usability of automated knowledge acquisition tools directly by a domain expert and assess the progress that has been made in current tools.

Two knowledge acquisition tools are examined in depth from the aspect of their ability to be used directly by a domain expert with minimal assistance from a knowledge engineer. The first, KNACK, is a knowledge acquisition tool which is appropriate for producing expert systems for report generation tasks. The second, AQUINAS, is a workbench of tools and techniques for knowledge acquisition combined within an integrated framework. The paper will describe our experience with these knowledge acquisition tools and assess the progress that has been made in respect to the objective of reducing the role of the knowledge engineer.

Finally, we will explore the general question of whether the use of knowledge acquisition tools will permit the domain expert to use the tool directly without intervention by a knowledge engineer. How close have we come to eliminating the knowledge engineer from the knowledge acquisition process?

INTRODUCTION

In constructing a knowledge-based system, one of the most difficult and time-consuming tasks is the elicitation and modeling of domain knowledge from a human expert. A commonly proposed solution is the development of automated tools for knowledge acquisition that interview a domain expert, analyze the knowledge acquired, and produce a knowledge-base to model the problem-solving expertise of the human expert. Unfortunately, these automated tools are often fairly complex and require the skills of a knowledge engineer for their use. Often, a knowledge engineer must interview the domain expert and interact with the tool since knowledge of artificial intelligence languages or concepts is often required. Yet the goal of many of these tools is to eventually be usable directly by a domain expert without reliance on a knowledge engineer.

This paper is concerned with the issue of the usability of the tools by the domain expert, and it focuses on limitations, recent progress, and proposals for future enhancements to current knowledge acquisition tools. Many of these knowledge acquisition tools are intended to be used directly by a domain expert without prior programming experience. These tools include AQUINAS (Boose and Bradshaw, 1987, Boose, 1988), SALT (Marcus, 1987, 1988), MOLE (Eshelman, 1988; Eshelman et al., 1987), MORE (Kahn et al., 1987), KNACK (Klinker, 1988, 1987), OPAL (Musen, 1988) and others. In this paper, we will examine in depth the issues involved in the usability of these automated tools by the domain expert.

A DOMAIN EXPERT'S VIEW OF TWO KNOWLEDGE ACQUISITION TOOLS

We will describe our experience with two of these tools, KNACK and AQUINAS. AQUINAS is a knowledge acquisition workbench of tools and techniques combined in an integrated framework. AQUINAS is a relatively mature tool derived from ETS, the Expertise Transfer System (Boose, 1986), a system in use at Boeing since 1983. AQUINAS provides a powerful and innovative array of knowledge acquisition tools and techniques. AQUINAS contains many features that enable some direct interaction with a domain expert. KNACK, developed more recently, is appropriate for developing expert systems for report generation tasks. KNACK, although less mature than AQUINAS, is a good example of a tool which is evolving towards eventual use by a domain expert.

Using KNACK to Construct Expert Systems: A Domain Expert's Experience

KNACK is a knowledge acquisition tool that specializes in acquiring knowledge for expert systems which perform document generation tasks. KNACK was originally developed at Carnegie Mellon University (Klinker, Bentolila et al. 1987) and was used to develop a knowledge-based system ("wringer") for the design of electromechanical systems and evaluation of a design in respect to nuclear hardening constraints. Since then, KNACK has been used as a knowledge acquisition tool in a variety of domains, including business plan generation, computer network design and configuration, electromechanical systems design documentation, and project progress reports. At Boeing, we have applied the KNACK knowledge acquisition tool to develop expert systems in two major problem areas: software requirements document production, and the generation of reports for independent research and development.

Developing a Software Requirements Document Wringer with KNACK. Our first experience in the use of the KNACK knowledge acquisition tool was to develop an expert system (wringer) to assist an analyst with software requirements documentation shown in figure 1. The initial version of the software requirements documentation "wringer" was constructed at Carnegie Mellon University by the developers of KNACK (Klinker, 1987).

KNACK is intended to be used directly by the domain expert without intervention by the knowledge engineer (Klinker, 1988). To construct a knowledge-based system with KNACK, the user must be able to enter knowledge in the form of a sample report and supply a conceptual model of the domain. The entry of the conceptual model has proven particularly troublesome for domain experts. In the earlier versions of KNACK, the conceptual model was elicited by KNACK using a

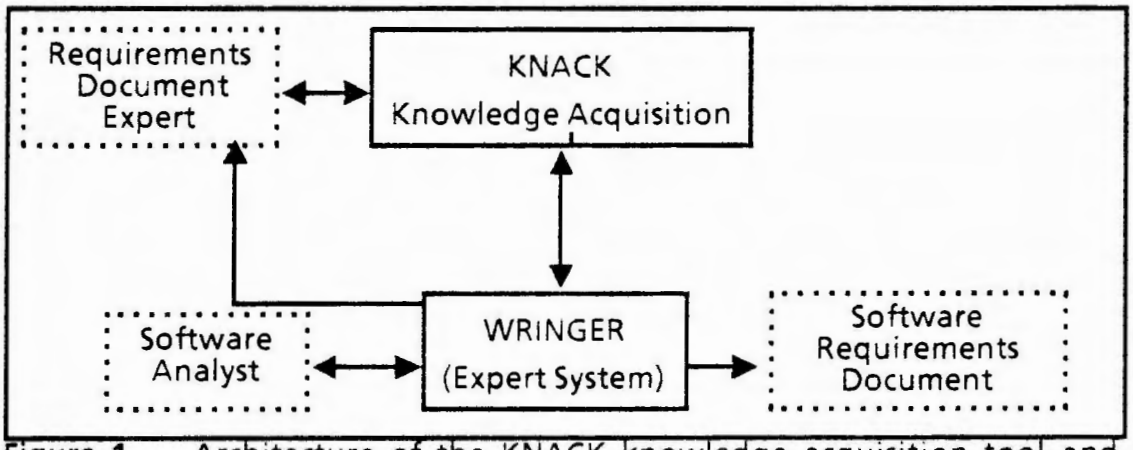


Figure 1. Architecture of the KNACK knowledge acquisition tool and software requirements documentation wringer.

question-and-answer dialog approach illustrated in figure 2 for the software requirements document generation expert system.

The major problem with this approach to capturing the domain model was that the question-and-answer style of entry was extremely tedious, cumbersome, and time-consuming. When we tried to replicate the entry of the domain model at Boeing using this technique in KNACK, we found that our domain expert was often confused by the fine distinctions in the terminology used in the dialog We felt strongly that the entry of the conceptual model could be improved from a standpoint of usability.

```
Loading the Domain Model...

How would you like to refer to the different environments in which systems
must function?
COMPANY, GROUP, SOFTWARE

What are the terms describing the characteristics of the SOFTWARE
environment?
PROJECT, AUTHOR, PURPOSE, DATE

How would you like to refer to conditions produced by the SOFTWARE
environment?
EXTERNAL FUNCTIONS

What are the terms you would like to use to refer to properties of an EXTERNAL
FUNCTION which do not have to be compared to condition properties?
SYSTEM, IDENTIFIER, VERSION, AUTHOR, TITLE, DATE
    :
```

Figure 2. The entry of concept and terms in KNACK using the original question-and-answer dialog approach.

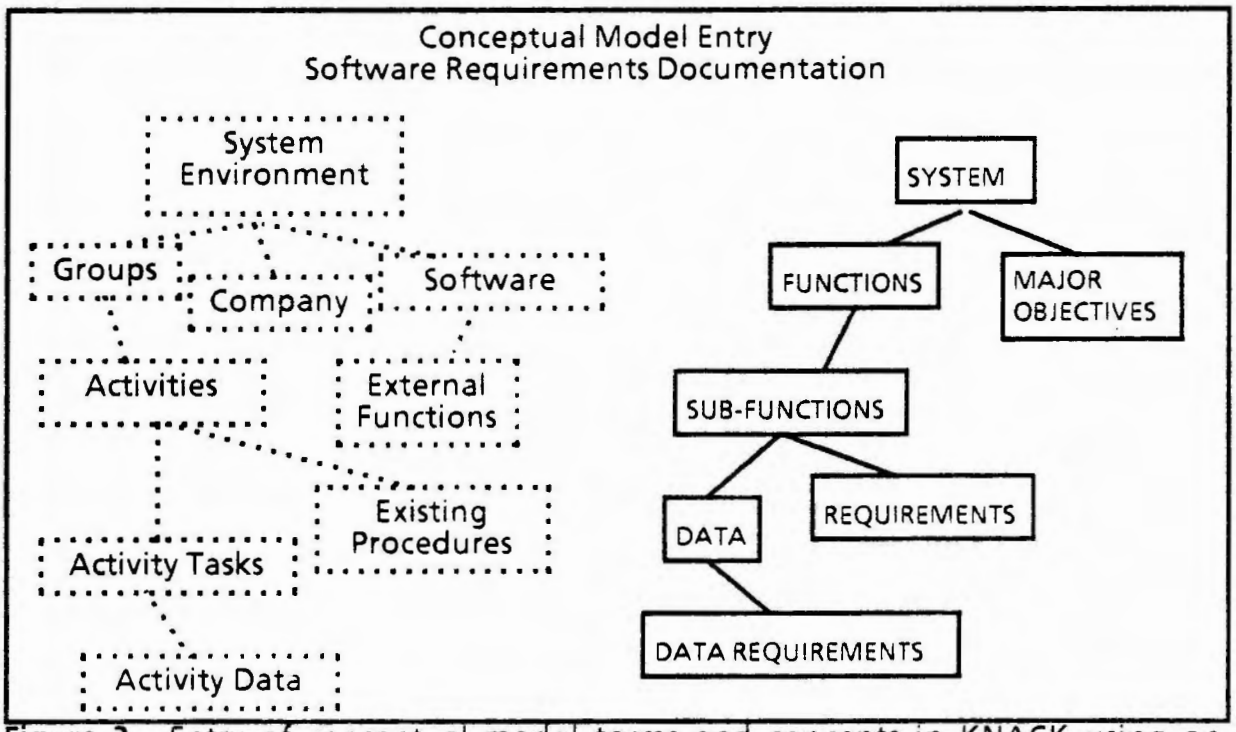


Figure 3 Entry of conceptual model terms and concepts in KNACK using an improved method of graphical input. Domain experts still require knowledge engineering assistance

KNACK has been recently modified to provide the capability for entry of the conceptual model using graphical input. The domain expert now selects options from a menu, and uses a mouse to define hierarchies of design and environmental domain model concepts. Although this process is a vast improvement over the question-and-answer approach, we have found that it still requires a knowledge engineer to work with the domain expert even after training. The domain expert still needs guidance in defining key concepts and their interrelationships. Figure 3 shows a portion of the conceptual model defined by our domain expert using the improved version of KNACK.

In the earlier version of KNACK, probably the most critical obstacle to its use by a domain expert was the neccessity for hand-coding enhancements to the KNACK-generated code to produce a usable wringer. Many features related to the user interface (menus, windows, help, explanations, hint windows, graphical techniques, and tabular input) were hand-coded to improve the software requirements documentation wringer. In addition, the inferences performed by the software requirements wringer were coded by a knowledge engineer since KNACK did not have the capability at that time to produce them. However, this is an example of how feedback from the domain expert on the poor usability influenced the evolution of KNACK. The current version of KNACK can produce a wringer that will run without requiring additional coding, although some important user interface features are still not automatically generated by KNACK.

Experience with the Improved KNACK: A Report Generation Task. The improved version of KNACK currently is being used to create an expert system (wringer) which performs a report generation task to produce research and development reports.

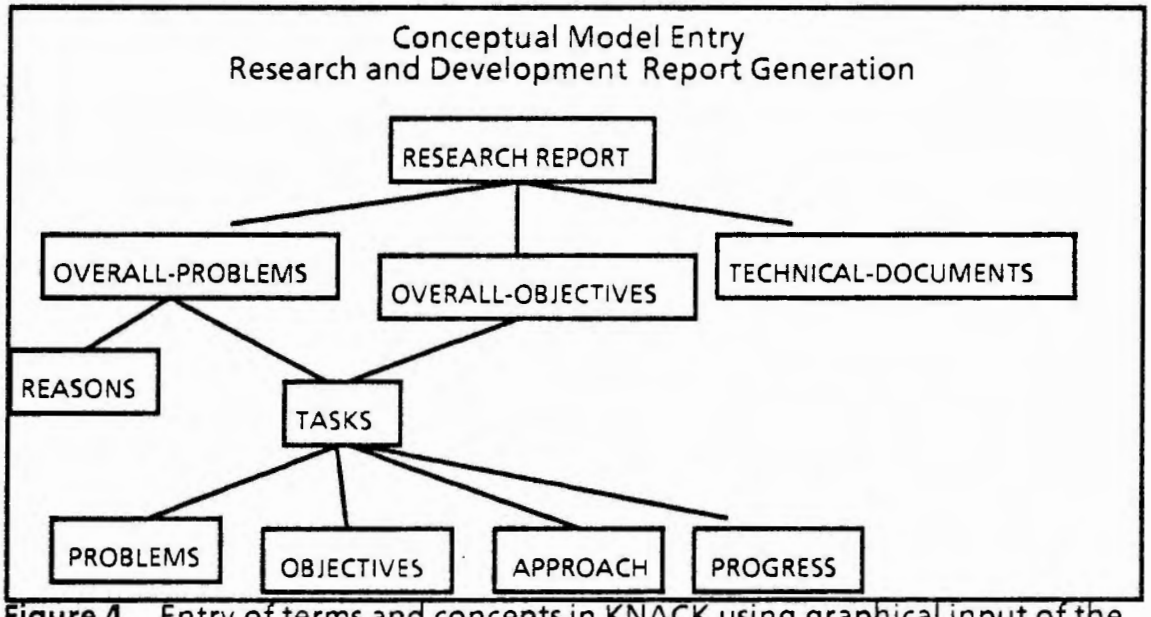


Figure 4. Entry of terms and concepts in KNACK using graphical input of the Research and Development Report Generator domain model

We used the graphical input techniques to define a conceptual model illustrated in figure 4. We have concluded that this graphical method of entry is much faster, but domain experts still require counsel.

We have found that experts sometimes have difficulty selecting a representative sample report and request help from the knowledge engineer. The report should contain all elements of the basic document structure, all boilerplate should be present, and the report should reflect any documentation standards and conventions. However, a greater source of confusion for the domain expert is how

1.0 Problem

The primary issue addressed by this project is the volume of logistic support plans required to support major system contracts.

Some problems that result from this are the following:

planning consumes many man-months of valuable engineering manpower,

the resulting plans have serious omissions,

the plans often have poor readability.

2.0 Objectives

The overall multiyear objectives for this project are to: (1) provide expert ai systems for logistics planning, (2) to develop computer advisory systems to guide in support equipment selection.

The specific objective for 1988 is to establish the economic and technical feasibility of developing effective expert ai systems for logistics planning, and support equipment selection.

Figure 5. A Sample Report for Research and Development Report Generation.

to select the relevant pieces of the sample report (fragments) to be input to the KNACK sample report editor. It is not necessary to enter the sample report in its entirety. A knowledge engineer familiar with how KNACK operates can quickly learn how to recognize report fragments which represent concepts, characteristics, specific examples of characteristics, boilerplate, or structural headings. Figure 5 shows a part of the sample research and development report.

We also learned that domain experts may not be familiar with the operation of expert systems or computer user interfaces. They may be confused by the options for entry of sample strategies, and they may not make good decisions about when knowledge should be inferred or calculated rather than requested from the user or when menus or graphical input might be preferable to a question. The expert tends to rely on the knowledge engineer for advice.

Finally, a common mistake is for the domain expert to enter examples in different formats in the sample report and the characteristics table for conceptual model terms. Figure 6 depicts the entry of concept characteristics and examples in KNACK for the research and development report writer domain. One difficulty is that phrases entered as example characteristics must be identical to phrases in the sample report.

Concept. Overall-Problems		
	Characteristics	Examples
Char1	Primary-issue	the volume of logistics support plans required to support major system contracts
Char2	Resulting-problems	planning consumes many man-months of valuable engineering manpower, the resulting plans have serious omissions, the plans often have poor readability

Figure 6. The entry of concept characteristics and examples in KNACK for the research and development report writer domain using tabular form.

Future Improvements for a More Usable KNACK. We are currently examining ways in which KNACK can be extended to improve its usability by the domain expert and reduce reliance on the knowledge engineer. A frequently heard comment from potential users of KNACK is that a tutorial or guidance facility is needed and will be crucial to the industrial use of KNACK by non-AI programmers. We are considering developing an introductory tutorial for domain expert users of KNACK. In addition, we are designing an interactive guidance facility to monitor the domain expert's use of KNACK and to provide assistance with specific tasks.

We are developing aids to help the domain expert use KNACK to create expert systems for documentation tasks. One area which is critical to the success of KNACK is the capability to display the sample report and the conceptual model simultaneously. The user should be able to highlight phrases in the sample report

and mouse on the graphical domain model to indicate a correspondence between a phrase and a concept or example characteristic in the domain model. A mechanism to perform simple pattern matching should identify similar entries in the sample report and conceptual model which do not correspond exactly and allow the expert to indicate if the two entries refer to the same concept or example.

A common source of confusion for most domain experts is creating the conceptual model. Experts are unsure of which terms in their domains constitute "concepts", which are "characteristics", and which are specific examples (instantiations) of characteristics. We are constructing a "library" of sample conceptual models from several document generation applications which the domain expert may display. The domain expert will be able to simultaneously view a segment of a sample report, the corresponding portion of the graphical domain model, and a tabular display of characteristics or constraints for a highlighted concept in some example domain.

We are also investigating an approach in which the domain expert first enters a sample report to a report preprocessor which helps the expert analyze the sample report to classify its various sections. Possible categories include boilerplate or fixed text (expected to appear in all examples of the report); superfluous phrases, words, or sentences; key concepts or terms in the specific documentation for which the wringer is to be created; characteristics or properties of the key concepts; specific instantiations or examples of characteristics which may vary from report to report or within the same report; instances in which a list of items or examples is expected or allowed; and optional phrases or examples. The preprocessor will then extract the essential pieces of the sample report that can then be loaded into KNACK. The knowledge obtained about the nature of the report fragments could then be used to provide guidance to the expert for conceptual model and strategy entry.

Experience with AQUINAS and Domain Expert Usability

AQUINAS is a knowledge acquisition workbench consisting of an integrated set of tools for knowledge acquisition, representation, and reasoning (Boose and Bradshaw, 1987; Boose, 1988). These tools include the Expertise Transfer System (Boose, 1986), which uses an interviewing methodology based on Personal Construct Psychology (Kelly, 1955; Gaines and Shaw, 1986) to elicit and refine knowledge from a domain expert. A variety of other tools and techniques are available to allow experts to decompose and structure knowledge in hierarchies, select a method for dealing with uncertain information, mix numeric and symbolic data types, and combine knowledge from multiple sources of expertise. This extensive mix of tools provides a powerful means for knowledge-base development, but the large number of options often proves confusing for the novice AQUINAS user.

Currently in AQUINAS, there are more than 250 options for expanding and refining knowledge once the initial knowledge base of solutions and problem-solving traits has been acquired. Although the workbench provides an extremely powerful and flexible set of capabilities, this flexibility is a deterrent to our objective of creating knowledge acquisition tools that are used directly by a domain expert. The expert may be confused or overwhelmed at the number of options available.

One source of confusion is that the expert may choose from several methods of reasoning, including rules with certainty factors, a maximum entropy approach, and the analytic hierarchy process. However, how many domain experts (or even

knowledge engineers) have the background to select the most appropriate method for their problem domain?

A maximum entropy approach might be appropriate for a diagnostic task in which the cost of obtaining information is very high. For example, in electronic or electromechanical diagnosis, certain diagnostic tests to detect faults may be expensive; in medical diagnosis, laboratory tests or diagnostic procedures may be costly or stressful on the patient. With a maximum entropy approach, only questions that yield the most distinguishing information are asked.

Progress in AQUINAS Toward Direct Use by a Domain Expert. One feature of AQUINAS that has contributed significantly to its usability by a domain expert is the AQUINAS tutorial. Originally developed to train knowledge engineers in a knowledge acquisition class, the tutorial helps domain experts become familiar with knowledge acquisiiton and refinement techniques in AQUINAS.

An important step toward making knowledge acquisition tools more usable directly by a domain expert is the development of guidance facilities to monitor the knowledge acquisition process A subsystem of AQUINAS called the dialog manager (figure 7) provides guidance in the use of AQUINAS and the knowledge acquisition process (Kitto and Boose, 1986, 1987). The advice-giving approach used by the dialog manager integrates learning with the actual use of AQUINAS, making it possible for the domain expert to "learn-while-doing", an approach found to more effectively motivate users than the drill and practice style of on-line tutorials or the often verbose help facilities (Carroll and McKendree, 1987). The dialog manager operates in a mixed initiative environment, offering several levels of detail in its suggestions and explanations. The dialog manager is based on a set of heuristics dealing with the use of AQUINAS and the knowledge acquisition process. These heurisitics are stored in the form of rules in the dialog manager's knowledge base. The dialog manager guides the user by analyzing the current state of the user's knowledge, the history of the user's actions and preferences, problems detected in rating grid knowledge, and complexity of the knowledge. The dialog manager suggests the next step to be performed in knowledge base expansion and refinement.

The dialog manager is even able to assist the domain expert in classifying an application task, identifying the appropriate problem-solving method, and suggesting the knowledge acquisition tools and strategies within AQUINAS to be used (Kitto and Boose, 1988). Application tasks can be classified according to various schemes (Hayes-Roth et al., 1983; Clancey, 1986). For a particular type of application task, generic problem-solving methods can be identified (Clancey,

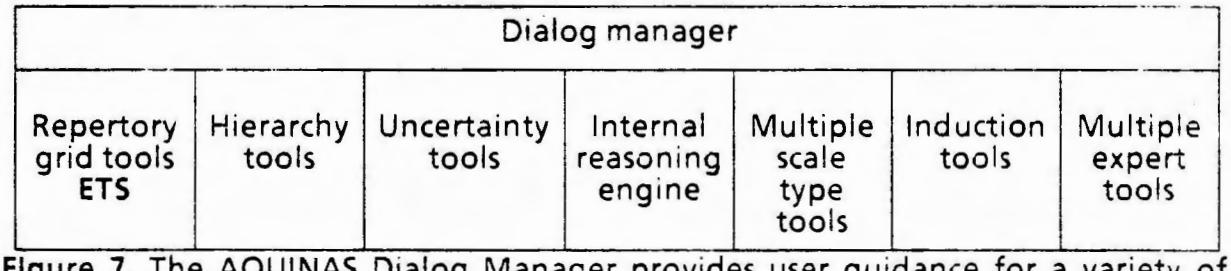


Figure 7. The AQUINAS Dialog Manager provides user guidance for a variety of knowledge acquisition tools.

1986), which can in turn be associated with specific knowledge acquisition methodologies (Bylander and Chandrasekaran, 1987). The Dialog Manager considers the application characteristics and problem-solving methods to select appropriate knowledge acquisition strategies within AQUINAS.

The common user interface in AQUINAS provides a consistent means of interacting between AQUINAS and the domain expert across the many techniques and tools offered in the AQUINAS workbench. Once an expert is familiar with the windows, menus, dialog manager, and graphical techniques in one part of AQUINAS, the identical interface will be encountered in the other AQUINAS tools.

AQUINAS also provides automated methods for knowledge-base refinement (Shema and Boose, 1988), including the capability to run batch test cases against modified rating grids (with adjusted rating values and trait weights) to improve performance of the grid. As a future extension to AQUINAS, we propose that a similar capability be added to compare test case results provided by the expert with the results obtained using different methods of reasoning and handling uncertain information. The reasoning mechanism that produced the results that most closely matched the test cases would be the technique recommended.

CONCLUSIONS

Even with the increasingly sophisticated knowledge acquisition tools available today, we are a long way from eliminating the knowledge engineer from the knowledge acquisition process. We have examined in depth two knowledge acquisition tools which are considered to be among the more powerful and innovative automated tools: KNACK and AQUINAS. We have found that much progress has already been made in respect to the usability of these tools by a domain expert with limited intervention by a knowledge engineer.

Advantages of KNACK for Domain Expert Usability

KNACK makes use of graphical displays, evaluation of knowledge for consistency and completeness, and explanations, prompts, help, and hint windows to provide guidance to a documentation domain expert. The improvements to KNACK currently underway will provide a tutorial/guidance/tutoring facility and special aids to analyze a complete sample report and extract those portions which are relevant to KNACK.

Advantages of AQUINAS for Domain Expert Usability

AQUINAS also contains numerous features which enhance its direct use by the domain expert. The AQUINAS tutorial has been used very successfully to train prospective knowledge engineers in knowledge acquisition classes at Boeing. The Dialog Manager subsystem in AQUINAS monitors the use of AQUINAS by the domain expert and suggests appropriate operations based on its heuristics dealing with problems and complexity in the knowledge base, user preferences and characteristics, and perceived benefits of the activity under consideration. A common user interface unites the many different tools and techniques in AQUINAS, providing a familiar and consistent mode of interaction with the expert. AQUINAS also has recently added consultation facilities which compare results obtained under AQUINAS with the results expected by the expert. We propose that these facilities could be extended to select a reasoning mechanism and methods of handling uncertainty.

Limitations to Domain Expert Use

However, in our experience with these tools, it is apparent that the knowledge engineer is still essential to the knowledge acquisition process. AQUINAS offers an overwhelming number of options for knowledge-base expansion and refinement, and the rating grid representation is usually unfamiliar to the domain expert. The tutorial and the Dialog Manager guidance facility offer help and enable more sophisticated domain experts to produce the initial rating grid knowledge, but a knowledge engineer is better equipped to take advantage of the powerful tools and techniques available in AQUINAS.

Similarly, KNACK also presents pitfalls to the domain expert who attempts to create a knowledge-base using KNACK Typically, the expert flounders in entry of the sample report, defining the conceptual model, and deciding the best stratgies for the expert system to acquire the knowledge it needs to produce the report. A knowledge engineer or AI programmer is still required today to enhance and tailor the code produced by KNACK to produce the final working wringer.

In both the AQUINAS and KNACK tools we see much progress in the development of features to improve usability by a domain expert. However, until a domain expert can actually sit down at a workstation and use an automated tool to develop a deliverable knowledge base without the frequent intervention of a knowledge engineer, we cannot claim that they are truly usable directly by the domain expert Despite many features and enhancements, the automated knowledge acquisition tools in use today still assure the knowledge engineer a role in the knowledge acquisition process in the near future.

ACKNOWLEDGEMENTS

The author wishes to thank Miroslav Benda, John Boose, Jeff Bradshaw, Jackson Brown, Georg Klinker, Sandra Marcus, Art Nagai, Dave Shema, and Sasha Smith for their contributions and support. AQUINAS and the Dialog Manager were developed at the Knowledge Systems Laboratory at the Boeing Advanced Technology Center of Boeing Computer Services in Seattle, Washington. KNACK was originally developed at Carnegie Mellon University and later enhanced at the Knowledge Systems Laboratory at the Boeing Advanced Technology Center.

REFERENCES

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

Boose, J.H. and Bradshaw, J.M (1987). Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems, *International Journal of Man-Machine Studies*, Volume 26, Number 1, pages 21-25.

Boose, J.H. (1988). Use of Repertory Grid-centered Knowledge Acquisition Tools for Knowledge-based Systems, special Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Bradshaw, J.M. and Boose, J.H. (1988). Decision Analytic Techniques for Knowledge Acquisition: Combining Solutions and Preference Models Using AQUINAS, special

Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Bylander, T. and Chandrasekaran, B. (1987). Generic Tasks in Knowledge-Based Reasoning: the 'Right' Level of Abstraction for Knowledge Acquisition, *International Journal of Man-Machine Studies*, Volume 26, Number 2, pages 231-244.

Carroll, J.M. and McKendree, J. (1987) Interface Design Issues for Advice-Giving Expert Systems, *Communications of the ACM*, Volume 30, Number 1.

Clancey, W.J. (1986). Heuristic Classification, in Kowalik, J.S (Ed.). Knowledge Based Problem Solving, pages 1-67, Edgewood Cliffs, N.J.: Prentice Hall.

Eshelman, L (1988). MOLE: A Knowledge Acquisition Tool that Buries Certainty Factors, special Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Eshelman, L., Ehret, D., McDermott, J., and Tan, M. (1987). MOLE: A Tenacious Knowledge Acquisition Tool, *International Journal of Man-Machine Studies*, Volume 26, Number 1, pages 41-54.

Gaines, B.R. (1988). Knowledge Acquisition: Developments and Advances in Oliff M.D., (Ed) Expert Systems and Intelligent Manufacturing, pages 410-434, New York: North Holland.

Gaines, B.R., and Shaw, M.L.G (1981). New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in Shaw, M.L.G. (Ed.) Recent Advances in Personal Construct Technology, New york: Academic Press.

Gruber, T. (1988). Acquiring Strategic Knowledge from Experts, special Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Hayes-Roth, F., Waterman, D., and Lenat, D.B (1983). Building Expert Systems, Reading, Mass: Addison-Wesley Press.

Jacobson, C. and Freiling, M.M. (1988). ASTEK: A Multi-Paradigm Knowledge Acquisition Tool for Complex Structured Knowledge, special Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Kahn, G., Breaux, E.H., Joseph, R.L., and DeKlerk, P. (1987). An Intelligent Mixed Initiative Workbench for Knowledge Acquisition, *International Journal of Man-Machine Studies*, Volume 27, Number 2, pages 167-179.

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

Kitto, C.M. (1987). Knowledge Acquisition Tools for Different Problem-Solving Paradigms: Research at Boeing Computer Services, in G. Salvendy (Ed.), Cognitive Engineering in the Design of Human-Computer Interaction and Expert Systems, pages 515-522, Amsterdam, The Netherlands: Elsevier.

Kitto, C.M., and Boose, J.H. (1986). Heuristics for Expertise Transfer: the Automatic Management of Complex Knowledge Acquisition Dialogs, Proceedings of the Expert Systems in Government Symposium, October 22-26, 1986, pages 53-63. Washington D.C. : IEEE Computer Society Press.

Kitto, C.M. and Boose, J.H. (1987). Heuristics for Expertise Transfer: An Implementation of a Dialog Manager for Knowledge Acquisition, *International Journal of Man-Machine Studies*, Volume 26, Number 2, pages 183-202.

Kitto, C.M. and Boose, J.H. (1988). Selecting Knowledge Acquisition tools and Strategies Based on Application Characteristics, special Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Klinker, G. (1988). KNACK: Sample-Driven Knowledge Acquisition for Reporting Systems, in Marcus S. (Ed.) Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers.

Klinker, G., Genetet, S., and McDermott, J. (1987). Knowledge Acquisition for Evaluation Systems, Proceedings of the Second Knowledge Acquisition for Knowledge-based Systems Workshop, Banff, Canada, October, 1987.

Klinker, G., Boyd, C., Genetet, S., and McDermott, J. (1987). A KNACK for Knowledge Acquisition, in Proceedings of the Sixth National Conference on Artificial Intelligence, Seattle, Washington, July, 1987.

Klinker, G., Bentolila, J., Genetet, S., Grimes, M., and McDermott, J. (1987). KNACK - Report-Driven Knowledge Acquisition, *International Journal of Man-Machine Studies*, Volume 26, Number 1, Pages 65-79.

Marcus, S., editor. (1988) . Automating Knowledge Acquisition for Expert Systems, Kluwer Academic Publishers, in press, 1988.

Marcus, S. (1987). Taking Backtracking with a Grain of SALT, *International Journal of Man-Machine Studies,* Volume 26, Number 4, pages 383-398.

Marcus, S., Stout, J., and McDermott, J. (1988). VT: An Expert Elevator Designer that Uses Knowledge-Based Backtracking, AI Magazine, Volume 9, Number 1, Spring 1988, Menlo Park, Ca : AAAI.

Musen, M.A. (1988). Generation of Model-Based Knowledge Acquisition Tools for Clinical Trial Advice Systems, Department of Computer Science, STAN-CS-88-1194, Stanford University, Stanford, Ca.

Musen, M.A. (1988). Conceptual Models of Interactive Knowledge-Acquisition Tools, Proceedings of the European Knowledge Acquisition Workshop, EKAW 1988. Gesellschaft fur Mathematik und Datenverarbeitung mbH, Sankt Augustin: GMD.

Shaw, M.L.G. and Gaines, B.R. (1987). Techniques for Knowledge Acquisition and Transfer, *International Journal of Man-Machine Studies*.

Shaw, M.L.G. and Gaines, B. (1987). KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices, *International Journal of Man-Machine Studies,* Volume 27, Number 3.

Shema, D.B. and Boose, J.H. (1988). Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, in special Issue on the AAAI Second Knowledge Acquisition for Knowledge-based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Building Expert Systems with KNACK

Georg Klinker, Casey Boyd, David Dong, Jon Maiman, John McDermott and Robert Schnelbach[1],

Department of Computer Science

Carnegie Mellon University

Pittsburgh, Pennsylvania 15213

Abstract

KNACK is an automated, specialized knowledge acquisition tool to generate expert systems that assist with different classes of reporting tasks. KNACK has been used by knowledge engineers to produce a series of reporting systems. Whenever such a system revealed inadequacies, KNACK was used to improve it. If individual task requirements could not be realized with KNACK, the knowledge base was manually enhanced. This paper summarizes the experience gained and outlines the scope of the current KNACK tool. It characterizes the application systems built, relates data on KNACK's performance to the defined characteristics, identifies shortcomings, and suggests directions along which to extend KNACK.[2]

1. Introduction

Existing expert systems have proven that AI techniques can be used to solve a variety of knowledge intensive problems. But expert systems are time-consuming to develop and difficult to maintain. A key issue in developing any expert system is how to update its large and growing knowledge base. One approach to keep a large knowledge base maintainable is by organizing it according to the different roles that knowledge plays [Chandrasekaran 83, Clancey 83, Neches 84]. Based on this realization a variety of knowledge acquisition tools have been produced during the past years to overcome those development and maintenance problems.

Some knowledge acquisition tools try to automate the knowledge acquisition process. An automated knowledge acquisition tool typically interacts with domain experts directly. No knowledge engineer is necessary to translate the expert's knowledge into production rules. An automated knowledge acquisition tool further organizes the knowledge it acquires, and generates an expert system. The domain expert can also use it to test and maintain the program it generates. The critical feature of such a tool is, that a domain expert can use it without having to know about programming in general and specific AI

---

[1]Georg Klinker, David Dong, and John McDermott are now with Digital Equipment Corporation, DLB5-3/E2, 290 Donald Lynch Blvd., Marlboro, Massachusetts, 01752

[2]This research was sponsored by the Defense Nuclear Agency (DNA) and the Harry Diamond Laboratories (HDL) under contract DNA001-85-C-0027, the Bares Foundation, Boeing Computer Services (BCS), and Digital Equipment Corporation (DEC). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DNA, HDL, the Bares Foundation, BCS, or DEC.

techniques.

One approach to automated knowledge acquisition is specialized tools. Specialized tools derive their power by presupposing the problem-solving method of the expert systems they generate [McDermott 88, Gruber 87]. A problem-solving method establishes and controls the sequences of actions required to perform some task. This control knowledge dynamically defines the order in which subtasks have to be solved to perform the overall task. It also defines the kind of domain specific knowledge that is applicable within each step, thereby making explicit the different roles knowledge plays. The granularity of the problem solving method is determined by the requirement that the knowledge represented by a knowledge role can be applied without further control knowledge, i.e., the time at which that knowledge will be brought to bear is determined by its role. Thus, the problem solving method helps to identify and classify the domain knowledge. It makes the different roles knowledge plays in a task explicit and suggests ways to organize the knowledge base according to the knowledge roles. It further defines the way knowledge interacts during problem solving. Examples of automated, specialized knowledge acquisition tools are TEIRESIAS [Davis 82], ETS [Boose 84], MORE [Kahn 85], and MOLE [Eshelman 87] for selective tasks, and SALT [Marcus 88] for constructive tasks.

KNACK [Klinker 88] is an automated, specialized knowledge acquisition tool that can be used by domain experts to create expert systems that assist with different classes of reporting tasks. Reporting tasks are part of everyday life in some domains. They involve collecting data and presenting them in the form of a document to communicate facts, ideas, and decisions. Writing proposals and progress reports, documenting design decisions, and defining requirements for a product are only a few examples. Producing a good report is a complex and difficult task. Some of the problems involved are gathering the right data efficiently and presenting the information intelligently. Examples of KNACK-generated reporting systems are expert systems to produce technical documents for different stages in the design of electromechanical systems, an expert system to report on the requirements for a planned software system including an executive summary, expert systems to document the progress of a project, including a proposal writer, a progress report writer, and an activity list writer, and an expert system to present an investment opportunity.

Figure 1-1 presents an overview of the interdependencies between KNACK and its generated expert systems adapted from Dietterich's simple model of learning systems [Dietterich 82]: A domain expert uses KNACK to generate a knowledge base. The presupposed problem-solving method operates on the knowledge base, acquires additional information from a user, and produces a report. The knowledge base plus the problem-solving method form an expert system. The performance of the expert system provides feedback to the domain expert. Whenever a WRINGER reveals inadequacies the domain expert uses KNACK to improve it. If individual task requirements cannot be realized with KNACK, the knowledge base has to be manually enhanced.

A KNACK-generated expert system uses the acquire-and-present problem-solving method. The expert system identifies relevant pieces of information for writing a report. It chooses between known strategies that can be used to gather the relevant information and selects one piece of information to acquire next, together with an appropriate strategy. The expert system applies the selected strategy and integrates the acquired information with the existing information. This acquisition process goes on until the user quits or no additional information is needed. The system produces a report to document the acquired information.

When KNACK generates an expert system it presupposes and exploits the associated problem-solving

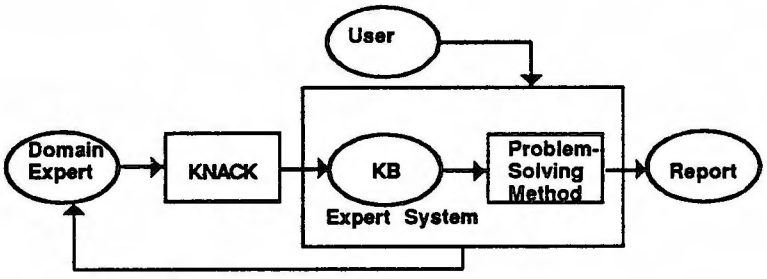


Figure 1-1: Overview

method. Explicit knowledge about an assumed problem-solving method and its associated knowledge roles defines what kinds of knowledge the expert must provide and when and how to apply the provided knowledge. This understanding of method and knowledge roles can be used during knowledge acquisition to guide a domain expert in defining, analyzing and testing a knowledge base.

However, using only the framework provided by the problem-solving method, the expert may be required to enter knowledge in a structured format that is unfamiliar to him. This imposes a learning burden on the expert. An important goal in the development of KNACK is to acquire knowledge from domain experts without presupposing knowledge engineering skills on their part. To achieve that goal KNACK exploits a domain model and takes a sample-driven approach to acquiring knowledge.

Like OPAL [Musen 88], KNACK exploits a domain model during knowledge acquisition. The domain model contains a structural and functional definition of a particular domain. It describes the concepts experts use and their interdependencies. Both, KNACK and OPAL, use the domain model to elicit knowledge in a format familiar to the expert and to develop expectations about the knowledge the expert might provide. The tools differ in the way the domain model can be customized for a particular domain. For OPAL, a knowledge engineer uses PROTEGE [Musen 88] to generate a customized knowledge editor like OPAL. KNACK tries to minimize the amount of knowledge engineering expertise required to build a domain model by customizing a model during an interaction with an expert.

KNACK uses a sample-driven approach to acquiring knowledge. This approach is summarized in figure 1-2. KNACK expects the expert to communicate a portion of his knowledge as a sample report and a domain model. The sample report is a document that exemplifies the report a KNACK-generated expert system is expected to produce. The domain model describes the concepts and the terms used in the sample report. During knowledge acquisition KNACK divides the sample report into small fragments and variabilizes each fragment by integrating it with the domain model, i.e., KNACK replaces specific terms in a fragment with the concepts they represent. To instantiate the identified variables for different applications, KNACK asks the expert for sample strategies and generalizes them. It displays several different instantiations of the generalized fragments and strategies, and the expert edits any of these that need editing. The corrections motivate and guide KNACK in refining the knowledge base. This process of abstraction and completion results in a knowledge base containing a large collection of generalized report fragments and strategies more broadly applicable than the sample report and sample strategies. Finally, KNACK examines the acquired knowledge for incompleteness and inconsistency.

A detailed description of KNACK can be found in [Klinker 88]. This paper does not introduce new

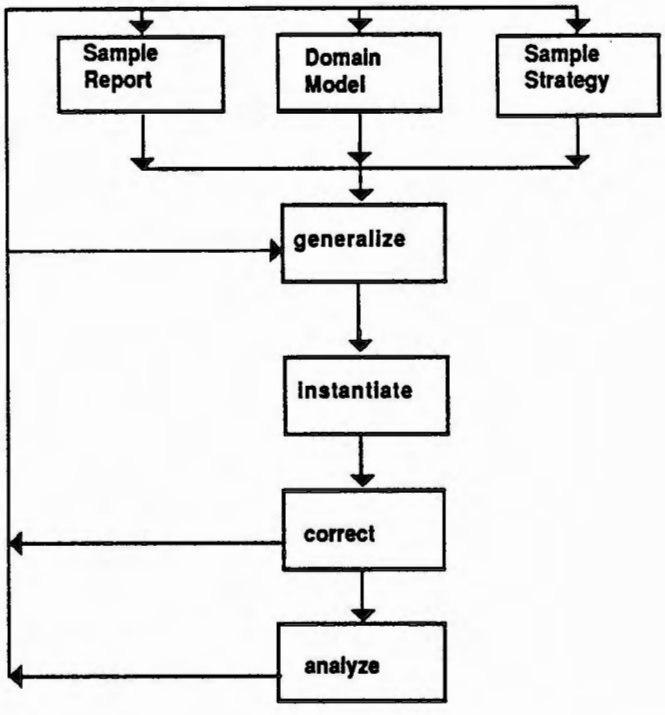


Figure 1-2: KNACK's Approach

features of the KNACK tool. It is meant to evaluate the existing tool. The following sections summarize the experience gained by knowledge engineers in using KNACK to build a series of application systems. Section 2 describes reporting tasks in general and outlines how KNACK's presupposed problem-solving method can be combined with other methods resulting in evaluative systems and design systems. Section 3 introduces the systems we have built with KNACK so far, characterizes them, relates data on KNACK's performance to the defined characteristics, identifies shortcomings, and suggests directions along which to extend KNACK. Section 4 gives some impression on the manual refinements of the generated knowledge bases to accommodate the individual requirements of a task. Section 5 defines the scope for the existing KNACK tool.

2. The WRINGERs

Reporting tasks exist as independent tasks or can be part of other problems. A variety of selection and design tasks need a considerable amount of information to begin with or require results to be documented. Improving the design of an electromechanical system to meet some external requirements and assisting with the definition of requirements for a planned software system are two examples. Existing expert systems and the associated knowledge acquisition tools concentrate their effort on the problem-solving process. They are not focused as much on obtaining preliminary information and presenting results. KNACK's simple acquire-and-present problem-solving method can be viewed as complementary to other problem-solving methods. For example, KNACK's method can be viewed as complementary to SALT's propose-and-revise systems.

SALT's propose-and-revise method proposes an extension to a partial plan or design and then checks for constraint violations. If a constraint is violated, the method finds the least costly fix that will eliminate the violation, and applies it. This process continues until no more additions can be made to the plan or design or no constraints are violated. The acquire-and-present problem-solving method serves as a front-end to acquire information and as a back-end to present results. Combining KNACK's and SALT's method broadens the scope of propose-and-revise systems to include design tasks that need a large amount of information to start with or require that a document be produced.

The method that combines KNACK's and SALT's method first uses appropriate interactive or autonomous strategies to gather necessary preliminary information representing an existing but partial plan or design. It then uses autonomous strategies to extend the partial plan or design, e.g., it infers values using domain-specific knowledge or formulas. For each plan or design extension, the method checks for constraint violations. If a constraint is violated, the combined method finds the least costly fix that will eliminate the violation, and applies it. This process continues until no more additions can be made to the plan or design or no constraints are violated. Finally, the method produces a report that documents the plan or design.

Depending on the kinds of knowledge provided by the expert, KNACK generates WRINGERs that dynamically decide which steps of the combined problem-solving method to use. This results in different problem-solving behavior of the generated expert systems creating, for example, design systems, evaluative systems, and reporting systems. A reporting task (as described previously) requires no knowledge defining constraint violations and remedies for violated constraints. Therefore, a WRINGER assisting with reporting tasks does not use the steps of the combined method that check for constraint violations or propose fixes for violated constraints.

Evaluation tasks require knowledge defining constraints but no knowledge of remedies for violated constraints. The purpose of an evaluation system is to decide whether to reject or accept a proposal. It identifies possible flaws in a plan or design, but suggests no fixes. A WRINGER assisting with evaluation tasks does not use the steps of the combined method that propose fixes for identified constraints. Examples of KNACK-generated evaluation systems are expert systems to evaluate designs of electromechanical systems from a hardening perspective, an expert system to identify problems with the requirements for a planned software system, and expert systems to evaluate the progress of a project from a management perspective.

Design systems require knowledge about constraint violations and associated remedies. Examples of KNACK-generated design systems are an expert system to improve the design of electromechanical systems that may be suboptimal from a hardening perspective, and an expert system to design computer networks.

3. Characterizing the WRINGERs

KNACK derives its power by exploiting the presupposed problem-solving method and the associated knowledge roles. A problem-solving method must be specific enough, so that it can be used during knowledge acquisition to guide a domain expert in defining, analyzing, and testing a knowledge base. On the other hand, a specific method limits the scope of an expert system that uses it. To get a better understanding of the kinds of tasks KNACK's assumed method can solve, knowledge engineers applied KNACK to the following domains:

- **The Data Item Description WRINGER Family**: Three WRINGERs have been developed for very

similar tasks. They assist with different stages in the design of electromechanical systems for the nuclear hardening domain. Nuclear hardening requires the use of specific engineering-design practices to increase the resistance of an electromechanical system to the environmental effects generated by the detonation of a nuclear weapon. Designers of electromechanical systems usually have little or no knowledge about the specialized analytical methods and engineering practices of the hardening domain. The purpose of the WRINGERs is to assist a designer in presenting a given design of an electromechanical system in such a way that hardening experts have readily available the information they need to evaluate the system from a hardening perspective. The first WRINGER, the Program Plan writer (PP WRINGER), produces the primary top-level report covering all phases of the design project. The second expert system, the Design Parameters Report writer (DPR WRINGER), improves and presents a detailed description of an electromechanical system ranging from the level of major components to the level of individual semiconductors. The last WRINGER, the Test Plan writer (TP WRINGER), presents a plan for confirming the hardness of a design. This includes a list of the design components to be tested, a description of the tests to be performed, and the expected test results.

- **The NAC WRINGER:** A WRINGER has been developed to assist a sales-person with the design and configuration of computer networks. In general, a salesperson has a good understanding of customer requirements involving cost, compatibility, and extendability when he is designing a network, but usually he has little or no knowledge about the detailed technical aspects of such a network. The WRINGER takes as input whatever the salesperson can provide, ranging from an incomplete list of high-level components (for example, a VAX-11/780 host, a VT100 terminal) to a completely specified network. The WRINGER then revises and completes that description as a technically functional computer network. The corrections and additions are either extensions to the design or interconnections between design components. The WRINGER's output is a list of generic network components and their interconnections, which serve as input for a program that will select the specific parts.

  For the NAC WRINGER, the actual implementation of the knowledge base and the problem-solving method differ from the implementation of the other expert systems. However, the format of the KNACK output is the same for all WRINGERs. A simple parser translates KNACK's output into OPS5 [Forgy 81] rules that correspond to the different implementations. Three rule types can be generated for both implementations (inference, menu, and constraint rules). Therefore, the NAC WRINGER's generated knowledge base is incomplete and has not been integrated with the real system. However, the generated rules successfully demonstrated the feasibility of this approach.

- **The Software Requirements WRINGER** (SR WRINGER): A WRINGER has been developed to assist a systems analyst with the definition of requirements for software. Defining requirements for new software is a very complex process. The WRINGER assumes that the user will describe the new software on a high level of abstraction. It supports the systems analyst in functionally decomposing that description into basic modules and defining the data requirements for the modules. The WRINGER checks for consistency (for example, internal input data for one module must be produced by another module) and compatibility (for example, the planned software must be integrated with the existing software environment). The WRINGER produces a technical document that describes the requirements for the software system and contains an executive summary with an opinion about whether the new software will be a valuable enhancement of the existing software.

- **The Project Progress Report WRINGER Family:** WRINGER expert systems have been developed to assist a project leader with the assessment of a project's progress. This includes a Proposal Writer, a Progress Report Writer , and an Activity List Writer. The Proposal writer (PPR1 WRINGER) acquires information about a planned project (for example, objectives, schedule, resources, tasks, project plan), points out inconsistencies (for example, objectives that cannot be achieved by the planned tasks, budget constraints), and produces a proposal. The Progress Report writer (PPR2 WRINGER) periodically acquires information about the project status (for example, budget, schedule, resources), compares the status to the actual project plan, and generates a progress report. The progress report emphasizes deviations from the project plan. Finally, the Activity List writer (PPR3 WRINGER) expands the tasks of a planned project into individual activities, relates them to persons working on the project, and

documents the progress and changes for the individual activities.

- **The Expert Marketing Methods Advisor:** A WRINGER has been developed to assist an entrepreneur in the preparation of marketing plans for a proposed business. An entrepreneur usually has little knowledge about how to create a marketing plan. The WRINGER assumes that the entrepreneur can provide a description of the product and the goals of the proposed business. It then advises the entrepreneur in creating the marketing plan. Its output is a document containing the necessary details and justifications to recommend the proposed business to an investor.

Many different mechanisms can be defined to characterize the above described WRINGERs. As is the problem with every taxonomy, the chosen mechanisms are subjective, seem arbitrary, and are by no means complete. This problem always remains, but can be reduced by choosing mechanisms with respect to a particular purpose and relating them to observable data. The mechanisms we chose are meant to highlight our experience gained in using KNACK to create the WRINGERs and to identify the scope of the existing KNACK tool. The WRINGERs can be characterized with respect to the following mechanisms:

- **Primary focus:** characterizes the different kinds of systems KNACK can create.
- **Sample report:** characterizes the reports a WRINGER produces.
- **Domain model:** characterizes the pieces of information specific to a WRINGER application.
- **Sample strategies:** characterizes the procedures a WRINGER uses to acquire information.
- **Constraints:** characterizes the constraints defined for a WRINGER's task.
- **Fixes:** characterizes the fixes defined to resolve violated constraints.

The results are summarized in Table 5-1. While creating the systems, data have been gathered to relate our experience with KNACK to the defined task criteria. This allows us to judge KNACK's suitability for the tasks it has been applied to, identify shortcomings, and suggest directions along which to extend KNACK. The following sections describe each of the criteria in detail.[3]

3.1. Primary Focus

The kinds of knowledge a WRINGER uses determines its primary focus: document, evaluate, or design. Reporting systems use the following kinds of knowledge:

- IDENTIFY RELEVANT INFORMATION: identifies pieces of information that may extend a partial report and organizes the required information around related topics.
- DEFINE A POSSIBLE STRATEGY: associates a set of strategies with each piece of information. Each strategy describes different ways to acquire that information.
- INTEGRATE ACQUIRED INFORMATION: describes how to integrate the newly acquired piece of information with the existing information.
- DEFINE THE SKELETAL DOCUMENT: determines the chapters, sections, and subsections that are part of a document, their order, and how to appropriately place fragments within the report.
- DEFINE A FRAGMENT: describes a small possible piece of the report.

Evaluative systems use in addition the following kind of knowledge:

---

[3]The SR WRINGER was created with a very early implementation of the KNACK tool. Quantitative data are not available. Therefore, the following will omit the SR WRINGER

- DEFINE A CONSTRAINT: defines restrictions for values of variables.

Constructive systems further require the following kind of knowledge:

- PROPOSE A FIX: suggests remedies for violated constraints and identifies the variables to be updated as a consequence of the remedy.

The kinds of knowledge a WRINGER uses determines the input the expert must provide. KNACK asks the user to input a domain model, a sample report, and sample strategies. Table 3-1 characterizes data on the input the expert had to provide to generate the different systems. This includes the sample report (number of fragments, words per fragment), the domain model (number of concepts, characteristics per concept, examples per characteristic, links, constraints, fixes), and sample strategies (number of questions, inferences, tables, menus, graphics, formulas). These kinds of input are not equally important for different tasks. Table 3-1 shows that the domain model is important for all WRINGERs. The domain model defines the pieces of information specific to an application and therefore the variables for a task. Some tasks primarily depend on the sample report, others on the sample strategies. The numbers reflect that strategies play a minor role for the PP, TP, NAC, and PPR3 WRINGER, and are a major factor for the DPR, PPR1, PPR2, and EMMA WRINGER. Table 3-1 shows further, that a significant part of the expert's input to create the WRINGERs is a sample report. Thus, the primary focus of all WRINGERs is to document. The only exception is the NAC WRINGER. Its purpose is not to document a task, but to evaluate and construct a plan or design. In addition to documenting, the DPR WRINGER's focus is also to evaluate and construct a plan or design. Table 3-1 shows that the DPR and NAC WRINGER are the only systems for which DEFINE A CONSTRAINT knowledge was provided. The PPR1 and PPR2 WRINGER each do use DEFINE A CONSTRAINT knowledge (see Table 4-2), but the rules were handcoded and not generated with KNACK. The DPR WRINGER is the only system with PROPOSE A FIX knowledge.[4]

Input to KNACK								
Input	PP	DPR	TP	NAC	PPR1	PPR2	PPR3	EMMA
Sample Report:								
# Fragments	92	140	136	-	124	132	32	45
# Words/Fragment	7.6	16.5	16.3	-	8.0	12.3	7.8	23.9
Domain Model:								
# Concepts	21	22	19	15	43	48	48	14
# Characteristics/Conc.	2.3	4.8	2.1	5.7	3.6	3.8	3.8	7.8
# Examples/Char.	1.6	1.4	1.5	1.5	0.96	0.95	0.95	0.7
# Links	19	30	17	80	42	47	47	13
# Constraints	-	1	-	49	-	-	-	-
# Fixes	-	4	-	-	-	-	-	-
Sample Strategies:								
# Questions	29	43	35	-	79	29	24	45
# Inferences	16	45	1	16	25	30	31	19
# Tables	-	-	-	-	-	-	-	-
# Menus	-	-	-	43	11	13	3	6
# Graphics	-	-	-	-	-	-	-	-
# Formulas	-	28	-	-	-	-	-	22

Table 3-1: Input

[4]For the NAC WRINGER the generation of PROPOSE A FIX knowledge has not been implemented.

KNACK was originally intended to produce only reporting systems. To investigate KNACK's applicability to other tasks, it was extended to generate evaluative and constructive systems. The assumption was, that reporting tasks exists as part of other problems and, therefore, KNACK's method can be viewed as complementary to other problem-solving methods. To demonstrate the feasibility of this idea, some of SALT's mechanisms for constructive tasks were integrated with the KNACK tool. KNACK can generate simple systems that produce a design or plan in a straightforward manner. Interacting and conflicting constraints and fixes are beyond its scope. The extended tool was tested to create the DPR and NAC WRINGER. Although successful, the use of KNACK for these tasks proved difficult. KNACK only provides a very simple mechanism to define DEFINE A CONSTRAINT and PROPOSE A FIX knowledge: a series of menus. This results in very general rules that almost always need to be refined. KNACK provides a simple rule editor for this purpose. The use of the rule editor does not require syntactic programming knowledge, but the expert must have an understanding of the interdependencies between the domain model and his input. This presupposes knowledge engineering skills on the expert.

To overcome the problems, KNACK's mechanism to define DEFINE A CONSTRAINT and PROPOSE A FIX knowledge could be improved by exploiting the domain model and integrating all of SALT's approach with KNACK. By exploiting the domain model in a similar manner as for generating fragment and strategy rules, KNACK could define the specific constraint and fix rules, making the rule editor superfluous.

3.2. Sample Report

The sample report mechanism characterizes the reports a WRINGER produces. This includes the following mechanisms:

- **Adapted or designed:** describes whether a sample report existed prior to creating a WRINGER or whether it had to be designed by the expert.
- **Structured or unstructured:** describes whether a report structure is common to a large set of applications.
- **Explicit or implicit:** describes whether a report typically reflects the knowledge about the task.

3.2.1. Adapted versus Designed

Some tasks typically include producing a document. Sample reports for specific applications exist or detailed guidelines on how to produce the document have been written. For these tasks existing reports and/or guidelines can be adapted by the expert and used as the sample report input for KNACK. If this is not the case, the expert has to design a sample report prior to an interaction with the KNACK tool.

The PP, DPR, and TP WRINGER's task is to produce a report according to detailed government guidelines. Sample documents existed. The nature of the EMMA WRINGER's task is to produce a document. Nevertheless, the effort to create the sample reports with KNACK was significant for these tasks as shown in Table 3-2. The table documents the effort involved in creating the WRINGERs. Effort is measured in terms of the expert's time spent to generate a WRINGER using KNACK. This includes the effort for the initial input (sample report, domain model, and sample strategies), the generalization process (sample report, sample strategies, constraints, and fixes), and the necessary manual refinements of the generated knowledge base. The effort for the generalizations includes the expert's corrections to the sample instantiations of the generalized fragments and strategies. The first line shows the total effort measured in number of days. Here, a word of caution might be appropriate. The WRINGERs were

created by knowledge engineers experienced in the use of KNACK and knowledgeable with the task requirements. A novice's effort will be significantly higher. The remaining lines of Table 3-2 document the percentage of the total effort devoted to the different stages in creating a WRINGER with KNACK. For a number of reasons, the actual data have to be taken with a grain of salt. They reflect the approximate time each of KNACK's functions was used by the knowledge engineer. The effort for the generalization process includes some of the effort to define the sample fragments and strategies. This is, because creating WRINGERs is an iterative process: Whenever a WRINGER reveals inadequacies, KNACK is used to improve it, i.e., problems discovered during the generalization process require to update the domain model, sample report, or sample strategies. Also, the numbers do not reflect the time required up-front to manually design the initial sample report, strategies, and domain model. This is a significant part of the expert's effort: The use of KNACK requires a well structured sample report and an understanding of the concepts used, their interdependencies, and associated strategies. Finally, the numbers reflect some additional effort due to the fact that KNACK is a tool currently under development and always changing. This is especially the case for the PPR WRINGERs.[5] Significant effort went into re-doing those WRINGERs with improved versions of KNACK. To sum up, Table 3-2 is meant to give some impression on the expert's relative effort in using the different functions of KNACK to create the WRINGERs.

Table 3-2 further indicates a decreased relative effort in producing the sample report for the PPR WRINGERs. This is somewhat misleading. Since the effort on manual refinements is significantly higher for those WRINGERs, the actual effort to create the sample report increased. This is due to the fact, that no sample report or guidelines existed. They had to be developed by the expert.

Effort [in % of Total Effort][6]								
Effort	PP	DPR	TP	NAC	PPR1	PPR2	PPR3	EMMA
Total Effort [# of days]:	4	13.5	5	4	27.75	37.5	10	50
Sample Report	25	15	40	-	11	21	10	25
Domain Model	19	19	15	38	21	11	5	20
Sample Strategies	12	30	10	25	16	5	10	2
Generalize Report	13	7	10	-	4	3	3	2
Generalize Strategies	6	4	5	12	3	1	2	2
Generalize Constraints	-	1	-	25	-	-	-	-
Generalize Fixes	-	2	-	-	-	-	-	-
Manually Refining KB	25	22	20	n.a.	45	59	70	49

Table 3-2: Effort

KNACK provides no help in designing a sample report, i.e., systematizing a task into a document. It up to the expert to actually define the report.

KNACK could be extended by providing a mechanism to design the report around the concepts defined in the domain model. This would be a first step to provide some help in systematizing a task into a document. However, the problem to define the right domain model remains. (For some ideas on how to

[5]The remaining part of the paper will refer to the PPR1, PPR2, and PPR3 WRINGER collectively as PPR WRINGERs whenever it is not necessary to distinguish between the individual systems.

[6]Since the generated rules were not integrated with the real system, manual refinements are not applicable for the NAC WRINGER.

improve the definition of domain models see section 3.3.) Ensuring that the sample report reflects the structure defined in the domain model further provides criteria in deciding whether a sample report is appropriate for the task at hand, i.e., to judge how good an actual sample report is.

3.2.2. Structured versus Unstructured

A sample report is defined by its structure (i.e., chapters, sections, subsections, etc.), fixed text components (i.e., text that will be printed as typed in by the expert), and variable text components (i.e., text specific to an application). A report can be structured or unstructured. For a structured document the report's structure is common to a large set of applications. Only variable text components are specific to an individual application. For some tasks a small set of highly structured reports exists or it is possible to systematize a task into a structured document. But this is not always the case. Some tasks require a large set of reports whose structure differs from application to application. The variable text components, fixed text components, and even the structure of a document might change with every application, resulting in unstructured reports.

The PP, DPR, TP, and PPR WRINGER's report structure is common to a large set of applications. For the EMMA WRINGER's task the sample report differs from application to application. It is highly unstructured. The effort to create the sample report is significantly higher.

KNACK provides some help in dealing with tasks that require unstructured sample reports. The expert can define different fragments that describe different situations for different applications by using different terms denoting known concepts. The generalization process conditions the fragments on the specific terms found. The resulting WRINGER will then produce a report with different fragments dependent on the actual application. Further, KNACK requires that the sample report is organized into chapters, sections, and subsections. Whole chapters, sections, and subsections are conditioned for particular applications if specific terms are found in the headings. As a result, a whole report part might or might not be part of the document produced by the WRINGER the expert is working to create. The above described techniques require deliberate use of variable text in chapter headings and other fragments where such use is sometimes inappropriate to the intent of he document. Mechanical sounding text presents a problem in a marketing plan, as it lessens the rhetorical quality needed to persuade the reader.

To make it easier to use KNACK for unstructured tasks, the tool could be extended to provide an explicit concept of alternatives and/or subreports.

3.2.3. Explicit versus Implicit

Some tasks are typically documented, i.e., the knowledge about the task is reflected by the report. For other tasks, only the results are reported, i.e., the document does not include the knowledge about the task. The PP, DPR, TP, and PPR WRINGER's task includes a document that explicitly describes how results were obtained. In contrast, the EMMA WRINGER's reports usually document results. How to derive results is implicit in the document. The domain model is essentially a definition of the variables which appear in the report. KNACK is less helpful when many of the variables are not in the report, but are used for intermediate processing. The only means KNACK has to acquire the task knowledge not represented by the sample report are strategies. This leads to an increased effort in creating the sample report for the WRINGER (see Table 3-2). A concept of indirect reports might extend KNACK's scope to include implicit reports. Indirect reports are not intended to be printed. They reflect the implicit knowledge necessary to obtain results and could be displayed during an interaction with a WRINGER. This would allow the user to follow the reasoning of the program more closely.

3.3. Domain Model

Other important task characteristics are reflected by the domain model. The model defines the pieces of information specific to an application. With respect to the model, the following mechanisms have been identified:

- **Adapted or designed:** describes whether a domain model was created prior to or during an interaction with KNACK.
- **Structured or unstructured:** describes whether a domain model can be described by a few or a lot of different node and link types.
- **Explicit or implicit:** describes whether the concepts and terms used in a domain model closely correspond to the pieces of information reported in the document.
- **Simple or complex:** describes how many concepts and terms are used in a domain model.
- **Constant or variable:** describes whether different applications use different instances of the same concept or different concepts.

3.3.1. Adapted versus Designed

The domain model can be defined prior to or during an interaction with KNACK. If it is possible to determine up front what the application specific terms of the sample report are, the domain model can be predefined. Generally, this is the case for tasks that can be represented by a structured report. Some tasks require the domain model to be defined iteratively during the generalization process. The generalization process integrates the domain model with the sample report, i.e., terms that are specific to an individiual application are replaced by the more general concepts denoted by the examples. For iteratively defined domain models inadequacies revealed during the generalization process make it necessary to update the domain model. This is often the case for tasks that require an unstructured sample report.

The PP, DPR, TP, NAC, and PPR WRINGER's tasks are well structured (as pointed out in section 3.2). As a consequence, it was possible to predefine most of the domain models for these WRINGERs. The increased effort for the NAC WRINGER (see Table 3-2) is due to the complexity of the model (see section 3.3.4). For the EMMA WRINGER, the domain model was defined interactively. This increased the effort to create the domain model.

KNACK provides a graphical editor to define the domain model in terms of nodes and links between nodes. Different node and link types are provided. The editor allows the expert to create and update the model easily if the model fits the predefined node and link types. However, updating the domain model does not automatically lead to an update of previous generalizations and instantiations. It is up to the expert to maintain consistency. Also, the mechanism to update the model in case the generalization process reveals deficiencies is somewhat awkward: The expert must quit the generalizer, invoke the domain model editor, update the model, quit the editor, and re-invoke the generalizer.

KNACK could be extended to accommodate better iteratively defined domain models. For example, a truth maintenance system could propagate changes in the model to previous generalizations and instantiations automatically. A graphical mechanism could be provided to refine the model from the sample report, i.e., the expert uses the mouse to point to fixed components in the sample report indicating that those should have been variable text components. This would update the model and result in the correct generalization.

3.3.2. Structured versus Unstructured

A domain model can further be structured, i.e., the concepts and terms can be expressed by only a few different node types and link types, or unstructured, i.e., a lot of different node types and link types are required. KNACK provides a small number of node types (Design Component, Environment Component, Design Connection Component, Environment Connection Component) and link types (Comprises, Connects To). They are predefined. This was sufficient for the PP, DPR, TP, NAC, and PPR WRINGERs. However the EMMA WRINGER required a domain model that could not be expressed completely with the predefined node and link types. Marketing plans are based on a description of changes over a period of time, such as business activity in calendar quarters over five years. However, the model does not represent time indexing of variables. Table 3-2 shows that significant effort went into unsuccessful trials to define the model. If the task fits the domain model editor's limitations, i.e., if it is well structured, KNACK can be easily used to define the model. However, KNACK provides no support for unstructured models. The expert cannot introduce new node and link types. This problem could be overcome by providing a mechanism for the expert to add new node types and link types as needed.

3.3.3. Explicit versus Implicit

The concepts and terms used in a domain model might explicitly correspond to the pieces of information reported in the document, i.e., the application specific pieces of information can be directly acquired via interactive strategies like questions and menus. Extensive use of autonomous strategies like inferences and formulas is characteristic of implicit models. An implicit model requires inference chains to arrive at a piece of information that are longer than 0. The length of an inference chain is determined by the number of autonomous strategies that establish preliminary information required to gather a particular piece of information. For example, the capacity of a terminal protection device can be inferred once its type is known. Since the type is determined via a question, acquiring the capacity requires an inference chain of length 0. On the other hand, vbulk of a cable shield can be computed once ibulk and dI/dtbulk of an outside cable and length, transfer resistance, and transfer inductance of the cable shield are known. Determining ibulk, dI/dtbulk, and transfer inductance requires an inference chain of 0. Acquiring length and transfer resistance requires an inference chain of 1. Therefore, determining vbulk requires an inference chain of 7. The PP, TP, NAC, and PPR WRINGER's domain models are explicit. For the PP and the TP WRINGER almost every piece of information can be acquired interactively from the designer using the question, table, or graphics strategies, i.e., no inference chain is required. The NAC and PPR WRINGERs use mainly menu or question strategies to acquire information. Their autonomous inference strategies mostly add information that depends on interactive strategies, i.e., the length of the inference chain is 0. By comparison, the DPR and EMMA WRINGERs use mostly autonomous strategies like formulas and inferences. It requires long inference chains to arrive at some pieces of information: Table 3-3 shows the length of the inference chains for the DPR WRINGER's autonomous strategies. As will be pointed out in section 3.4, it is difficult to define a large number of strategies with KNACK. A more sophisticated mechanism to define strategies is called for.

DPR WRINGER's Length of Inference Chain						
Length of Inference Chain	interactive	0	1-5	6-10	11-15	> 15
# Strategies	43	47	5	7	9	5

Table 3-3: Length of the Inference Chains for the DPR WRINGER's Autonomous Strategies

3.3.4. Simple versus Complex

The complexity of a domain model is determined by the number of concepts, characteristics, examples, and links defined. A domain model is simple if it uses some 20 concepts and less than 5 characteristics/concept. Table 3-1 shows that some 20 concepts are sufficient for the PP, DPR, TP, NAC, and EMMA WRINGERs. A domain model is complex if it contains more than 20 concepts and/or more than 5 characteristics/concept. The NAC and EMMA WRINGER's model include significantly more characteristics, examples, and links. This results in more complex models. As shown in Table 3-2, a more complex domain model - with respect to characteristics and examples - does increase the effort to define it with KNACK. This is a different story if the number of concepts increases. The PPR WRINGERs use significantly more concepts. The relative effort in creating the model does not change. The PPR3 WRINGER seems to be an exception. This is explained by the fact that its model is the same as for the PPR2 WRINGER and, thus, could be reproduced with little effort. KNACK provides some support for complex models. For example, the domain model editor allows the expert to move nodes and links around, display different parts of the model, zoom in or out to parts of the model, display different viewpoints of the model, etc. The editor could be improved by enhancing those mechanisms. Also, features should be added to display and update constraints and fixes.

3.3.5. Constant versus Variable

Finally, a model can be constant or variable. We call a model constant if different applications use different instances of the same concept. A variable model defines different concepts for different applications. The models for the PP, DPR, TP, PPR, and EMMA WRINGER are constant. Almost every concept is used by a particular application. Only the actual values are different. The NAC WRINGER requires a variable model. Each application combines different concepts into individual computer networks. KNACK provides no support for variable models. As a consequence, it is difficult to use KNACK to define the circumstances under which a concept will be used for different applications. KNACK's scope could be extended to include variable models by providing an explicit concept of alternative models and submodels.

3.4. Sample Strategies

Different WRINGERs use different combinations of strategies to acquire information. Strategies are either interactive or autonomous. Interactive strategies acquire information from a user. Examples are questions, tables, menus, or graphics. Autonomous strategies infer additional information based on already acquired information. Examples are inferences and formulas.

According to Table 3-1 the PP, TP, NAC, PPR, and EMMA WRINGERs use mostly interactive strategies like questions, tables, menus and graphics. The DPR WRINGER does use interactive strategies but mostly autonomous strategies like inferences and formulas.

This allows some insight into KNACK's suitability for the DPR WRINGER's task. Table 3-2 shows that the focus on strategies significantly increases the effort to create the DPR WRINGER. It is difficult to define a large number of strategies with KNACK. Strategies are highly interdependent: They use the values of multiple variables as input to determine additional values of other variables. Also, multiple strategies can use values of different input variables to determine values for the same variables. KNACK provides no support to keep track of those interdependencies. It is up to the expert to define a consistent set of strategies. Interactive strategies are associated with text, e.g., the actual text of a question, the text asking to choose among the items of a menu. They are generalized the same way the sample report is: by replacing terms that are specific to an individual application with the concepts denoted by the examples.

Autonomous strategies are a different story. There is no text associated with inferences and formulas. They are generalized by interpreting the relationships between concepts. KNACK uses some heuristics for this purpose. The heuristics work reasonably well for the DPR and NAC WRINGER's domain. Table 4-2 indicates that the generated rules do not have to be manually enhanced. But the same heuristics do not work at all for other problems, like the EMMA WRINGER's task. Here, a high percentage of inference rules are manually enhanced or entirely handcoded (see Table 4-2).

Making it easier to input strategies could significantly improve KNACK's performance and suitability for a variety of tasks. A more sophisticated mechanism to input strategies or graphical display and manipulation of interdependencies between strategies would be an important first step to overcome those deficiencies. This is especially important for autonomous strategies like inferences and formulas. Also, the heuristics used to generalize inferences and formulas should be more general applicable. Another solution might be to allow an expert to customize the heuristics for a particular task. Ultimately, KNACK could learn how to refine its heuristics from the expert's corrections of the instantiated generalizations.

3.5. Constraints

The constraint criteria defines the kind of constraints defined for a task. Constraints can be numerical tests, (i.e., the values of two numerical variables are described by a predefined relationship), set membership tests, (i.e., the value of a variable must be a member, could be a member, or cannot be a member of a predefined set of possible values), or required tests (i.e., a value of a variables requires that the value of some other variable is defined). In order for KNACK to be a suitable tool for evaluation systems, it must be possible for the user to identify constraints. As of now, this criteria seems to be somewhat artificial because of our limited experience in defining constraint knowledge with KNACK. Only the DPR and NAC WRINGER use constraints defined with KNACK (see Tables 3-1 and 3-2). They use constraints to define numerical tests and required tests. For the PPR1 and PPR2 WRINGER the constraint knowledge has been handcoded (see Table 4-2). The constraint criteria will become more and more important as more evaluative systems are being created with KNACK. It is obvious to us, that here is room for significant improvement.

3.6. Fixes

The fix criteria defines the kinds of fixes used. Fixes can be concepts, (i.e., a component is added to a design or plan), or strategies, (i.e., existing components are modified according to the specified strategy. It must be possible to identify at least one remedy for each constraint violation, if KNACK is to be used for constructive tasks. KNACK further assumes that the expert can provide an ordering for fixes. Interacting and conflicting constraints and fixes are beyond its scope. The same reservations as in section 3.5 apply here. Only one system exists, the DPR WRINGER, that uses very limited fix knowledge (see Tables 3-1 and 3-2). That WRINGER uses concept fixes.

4. Manual Refinements

KNACK's shortcomings - described in the previous section - resulted in manual refinements of the generated knowledge base. The enhancements were necessary to accommodate the individual requirements of a task. Creating WRINGERs is an iterative process. Whenever a WRINGER revealed inadequacies, either KNACK was used to improve it or the generated knowledge base was manually enhanced. If problems discovered during the generalization process required the expert to update the domain model, sample report, or sample strategies, first KNACK was used to update the knowledge base.

If individual task requirements could not be realized with KNACK, the knowledge bases had to be manually enhanced. The manual enhancements uncovered deficiencies and shortcomings in KNACK's approach to acquire knowledge, assumed problem-solving method, and associated knowledge roles. KNACK has been improved to address many of the encountered problems and was then used to re-generate or update the initial systems. This process was repeated several times resulting in the described application systems.

Table 4-1 gives an impression on the size of the generated systems. It lists the number of generated rules for each rule type and WRINGER. Report rules correspond to DEFINE A FRAGMENT knowledge; skeleton rules correspond to DEFINE THE SKELETAL DOCUMENT and IDENTIFY RELEVANT INFORMATION knowledge; question, inference, table, menu, graphics, and formula rules correspond to DEFINE A POSSIBLE STRATEGY and INTEGRATE ACQUIRED INFORMATION knowledge; constraint rules correspond to DEFINE A CONSTRAINT knowledge; fix rules correspond to PROPOSE A FIX knowledge. Line "Autonomous Rules" shows the number of rules that gather a piece of information without interaction with a user. Autonomous rules are inference rules, formula rules, constraint rules, and fix rules. The numbers in parentheses list the percentage of the knowledge base that are autonomous rules.

Generated Knowledge Base								
Rule Type	PP	DPR	TP	NAC	PPR1	PPR2	PPR3	EMMA
# Rules generated	288	602	383	285	531	469	183	239
# Report Rules	173	278	279	-	270	300	54	45
# Skeleton Rules	25	83	32	-	20	12	10	4
# Question Rules	58	86	70	-	158	58	48	90
# Inference Rules	32	90	2	64	50	60	62	38
# Table Rules	-	-	-	-	-	-	-	-
# Menu Rules	-	-	-	172	22	26	6	18
# Graphics Rules	-	-	-	-	-	-	-	-
# Formula Rules	-	56	-	-	-	-	-	44
# Constraint Rules	-	1	-	49	-	-	-	-
# Fix Rules	-	8	-	-	-	-	-	-
Autonomous Rules	32 (11)	155 (26)	2 (0.5)	113 (40)	50 (9)	60 (13)	62 (34)	82 (34)

Table 4-1: Generated Knowledge Base

Not all problems with KNACK's generated code have been resolved. Some of the remaining problems are listed in this section. The intention is to give some insight on the kinds of necessary manual refinements and not to provide a complete "bug" list. The manual refinements include rules that were completely handcoded and corrections of specific rules. The effort in enhancing the knowledge base is shown in Table 4-2. It lists the number and percentage of rules that had to be manually enhanced or handcoded for each rule type. Percentages are shown in parentheses. The first line lists the number of rules in the knowledge base. This includes the number of rules generated and the number of rules handcoded. The lines "Rules enhanced" and "Rules handcoded" show the number and percentage of rules in the knowledge base that had to be manually enhanced or completely handcoded, respectively. The remaining lines document the number and percentage of rules for each rule type that had to be enhanced or handcoded.

- KNACK cannot generate the rules encoding the graphics strategy. The corresponding rules for the PP, DPR, and TP WRINGER have to be handcodced. The same is true for the PP WRINGER's rules encoding the table strategy and the specific constraint rules for required tests for the PPR1 and PPR2 WRINGER.

- KNACK cannot generate the rules required by some specialized strategies. Those rules have to be manually enhanced (MIN function for 1 formula rule of the DPR WRINGER, choices depending on previously gathered information for 4 menu rules of the PPR2 WRINGER).

Manual Refinements of the Generated Knowledge Base[7]								
Rule Type	PP	DPR	TP	NAC	PPR1	PPR2	PPR3	EMMA
# Rules in KB	344	620	401	285	532	470	183	377
Rules enhanced	-	6 (1.0)	-	n.a.	53 (9.9)	51 (10.8)	20 (10.9)	170(45.1)
Report	-	3 (1.7)	-	n.a.	27 (10.0)	40 (13.3)	10 (18.5)	26 (35.6)
Skeleton	-	-	-	n.a.	10 (50.0)	5 (41.6)	5 (50.0)	4 (66.7)
Question	-	-	-	n.a.	16 (10.1)	6 (10.3)	5 (10.4)	73 (59.8)
Inference	-	1 (2.2)	-	n.a.	-	-	-	37 (52.8)
Table	-	-	-	n.a.	-	-	-	-
Menu	-	-	-	n.a.	-	4 (10.2)	-	8 (44.4)
Graphics	-	-	-	n.a.	-	-	-	-
Formula	-	1.8 (1)	-	n.a.	-	-	-	22 (25.0)
Constraint	-	-	-	n.a.	-	-	-	-
Fix	-	-	-	n.a.	-	-	-	-
Rules handcoded	56 (16.3)	18 (2.9)	18 (4.5)	n.a.	1 (0.2)	1 (0.2)	-	138(36.6)
Report	-	-	-	n.a.	-	-	-	28 (38.4)
Skeleton	3 (10.7)	1 (1.1)	1 (3.0)	n.a.	-	-	-	2 (33.3)
Question	-	-	-	n.a.	-	-	-	32 (26.2)
Inference	-	-	-	n.a.	-	-	-	32 (45.7)
Table	36 (100)	-	-	n.a.	-	-	-	-
Menu	-	-	-	n.a.	-	-	-	-
Graphics	17 (100)	17 (100)	17 (100)	n.a.	-	-	-	-
Formula	-	-	-	n.a.	-	-	-	44 (50.0)
Constraint	-	-	-	n.a.	1 (100)	1 (100)	-	-
Fix	-	-	-	n.a.	-	-	-	-

Table 4-2: Manual Refinements of the Generated Knowledge Base

It is obvious that KNACK needs to be extended to define the additional strategies and to generate the required more specialized rules.

5. Conclusion

Table 5-1 summarizes the results of the previous section and defines KNACK's scope. The existing KNACK tool is applicable for tasks whose primary focus is to document. It must be possible for the expert to provide a sample report, domain model, and sample strategies. KNACK is most appropriate for tasks which in addition can be characterized as follows:

- The sample report can be adapted or designed, and must be structured and explicit.
- The domain model must be adapted, structured, explicit, simple, and constant.
- The sample strategies must be mainly interactive.

[7]Since the generated rules were not integrated with the real system, manual refinements are not applicable for the NAC WRINGER.

The above criteria are met for the PP, DPR, TP, and PPR WRINGER's task.

However, KNACK needs to be improved to make the definition of sample strategies easier for the expert. A more sophisticated editor for sample strategies is called for. This will reduce the effort to create the DPR WRINGER. Also, the experience of the knowledge engineers to create the PPR WRINGERs shows, that designing the sample report and the domain model constitutes a major part of the expert's task. Here, KNACK's support could be improved by extending the features of the domain model editor and providing a mechanism to structure the report around the concepts in the domain model.

Using KNACK to create the EMMA WRINGER proved to be difficult. For the EMMA WRINGER's task, the sample report and domain model are unstructured and implicit. Extending KNACK's scope to include that task would be an extensive undertaking. It involves to improve the sample report mechanism (i.e., provide an explicit concept of alternatives and/or subreports and indirect reports) as well as the domain model mechanism (i.e., allow to introduce new node and link types, make the definition of autonomous sample strategies easier).

Currently, KNACK's applicability to evaluative and constructive tasks is very limited. KNACK's performance in creating the DPR and NAC WRINGER demonstrates that the tool could be extended to include evaluative and constructive systems. It could exploit the domain model in a similar manner as for generating fragment and strategy rules to assist the expert in defining the DEFINE A CONSTRAINT and PROPOSE A FIX knowledge. Using KNACK, the expert would not have to define the preconditions for constraint and fix rules as required by SALT. On the other hand, integrating all of SALT's approach with KNACK would lead to a duplication of the SALT effort. A more meaningful objective for future work would be to investigate how KNACK and SALT can be combined into a tool that recognizes reporting and constructive tasks and invokes the appropriate knowledge acquisition strategies. The combined tool would broaden the scope of propose-and-revise systems to include constructive tasks that need a large amount of information to start with or require a document to be produced.

Acknowledgments

We would like to thank Larry Eshelman and Don Kosy for many helpful discussions on how to define mechanisms to characterize the tasks of the different WRINGERs.

References

[Boose 84] Boose, J.
Personal construct theory and the transfer of human expertise.
In *Proceedings of the 4th National Conference on Artificial Intelligence.* Austin, Texas, 1984.

[Chandrasekaran 83] Chandrasekaran, B.
Towards a taxonomy of problem solving types.
AI Magazine 4(1):9-17, 1983.

[Clancey 83] Clancey, W.
The advantages of abstract control knowledge in expert system design.
In *Proceedings of the 3rd National Conference on Artificial Intelligence.* Washington, D.C., 1983.

Task Characteristics								
Characteristic	PP	DPR	TP	NAC	PPR1	PPR2	PPR3	EMMA
Primary Focus	document	document evaluate construct	document	evaluate construct	document	document	document	document
Sample Report	adapted structured explicit	adapted structured explicit	adapted structured explicit	- - -	designed structured explicit	designed structured explicit	designed structured explicit	designed unstructured implicit
Domain Model	adapted structured explicit	adapted structured implicit	adapted structured explicit	adapted structured explicit	adapted structured explicit	adapted structured explicit	adapted structured explicit	designed unstructured implicit
	simple constant	simple constant	simple constant	complex variable	complex constant	complex constant	complex constant	complex constant
Sample Strategy	question table graphics	question inference graphics formula	question graphics	inference menu	question inference menu	question inference menu	question inference menu	question inference menu formula
Constraint	-	num.test	-	num.test required	required	required	-	-
Fix	-	concept	-	-	-	-	-	-

Table 5-1: Task Characteristics

[Davis 82] Davis, R. and D. Lenat.
Knowledge-Based Systems in Artificial Intelligence.
McGraw-Hill, New York, NY, 1982.

[Dietterich 82] Dietterich, T.G.
Learning and Inductive Inference.
P.R. Cohen, and E.A. Feigenbaum (Eds.), The Handbook of Artificial Intelligence.
Morgan Kaufmann, 1982.

[Eshelman 87] Eshelman, L., D. Ehret, J. McDermott, and M. Tan.
MOLE: A Tenacious Knowledge Acquisition Tool.
International Journal of Man-Machine Studies 26(1):41 - 54, 1987.

[Forgy 81] Forgy, C.L.
OPS5 user's manual.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1981.

[Gruber 87] Gruber, T., and P. Cohen.
Design for Acquisition: Principles of Knowledge System Design to Facilitate Knowledge Acquisition.
International Journal of Man-Machine Studies 26(2):143 - 159, 1987.

[Kahn 85] Kahn, G., S. Nowlan and J. McDermott.
MORE: an intelligent knowledge acquisition tool.
In *Proceedings of Ninth International Conference on Artificial Intelligence.* Los Angeles, California, 1985.

[Klinker 88] Klinker, G.
KNACK: Sample-Driven Knowledge Acquisition for Reporting Systems.
S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.
Kluwer Academic Publishers, Boston, August 1988.

[Marcus 88] Marcus, S.
SALT: A Knowledge-Acquisition Tool for Propose-and-Revise Systems.
S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.
Kluwer Academic Publishers, August 1988.

[McDermott 88] McDermott, J.
Preliminary Steps Toward a Taxonomy of Problem-Solving Methods.
S. Marcus (Ed)., Automating Knowledge Acquisition for Expert Systems.
Kluwer Academic Publishers, August 1988.

[Musen 88] Musen, M.
Generation of Model-Based Knowledge Acquisition Tools for Clinical-Trial Advice Systems.
PhD thesis, Department of Computer Science, Stanford University, 1988.
STAN-CS-88-1194.

[Neches 84] Neches, R., W. Swartout, and J. Moore.
Enhanced maintenance and explanation of expert systems through explicit models of their development.
In *Proceedings of IEEE Workshop on Principles of Knowledge-based Systems.* Denver, Colorado, 1984.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

DRAFT

Learning Plans and Information Requirements for Pilot Aiding

(Draft)

Keith Levi
David Perschbacher
Valerie Shalin

Honeywell Systems and Research Center
Minneapolis, MN
May 1988

INTRODUCTION

We are developing an explanation based learning (EBL) system to learn plans for a large pilot-aiding expert system. The learning and plan recognition approach that we are pursuing should generalize to a wide variety of intelligent interface applications. The key features of the problem domain are that there is a human who is performing a procedural task (e.g., flying a plane, planning a strategic operation, designing a microchip, designing a building, working with a complex computer system, etc.) and an expert system that is supposed to assist the human. This problem domain presents a variety of technical challenges for automated knowledge acquisition, including learning about temporal relations, partial plan completion, and the schema activation issues that arise in planning with a plan library. We will also discuss issues in obtaining learning experiences from the performance system, inserting learned information into the performance system, and the requirements of the domain theory that is needed for the learning system. We presently have developed a preliminary planning/learning shell that is based on the TWEAK planning system (Chapman, 1985) and DeJong's EBL system (DeJong & Mooney, 1987). We are applying this shell to a large pilot-aiding expert system in a joint industry/university research effort involving Honeywell, Lockheed, Teknowledge, Search, and the Universities of Illinois and Michigan.

A Machine Learning Approach to Knowledge Acquisition

Machine learning techniques can be of great value for automating certain aspects of knowledge acquisition. Traditionally, the design of knowledge-based, expert systems has involved a process of interaction between a domain expert and a knowledge engineer. During this interaction, called *knowledge acquisition*, the knowledge engineer formalizes the expert's knowledge as inference rules and data structures. There are a number of obvious problems with this approach. First, developing an expert system can require an enormous amount of effort. A system may fail to perform properly in its application domain and require additional modification. Typically, this will involve continuing, ill-defined interactions between expert and engineer during which the system is gradually modified as performance failures arise. Second, although an expert may exist for a given domain there is no guarantee that the expert can adequately access and describe his or her own knowledge so that it can easily be represented in a program. Finally, as the first problem implies, it is difficult to determine when the knowledge base is correct, consistent, and complete. Knowledge acquisition techniques amount to a process of generate, test, and modify. A system is considered reliable when it performs well for some arbitrary period of time.

In view of the problems with current techniques, researchers have been interested in automating the process of knowledge acquisition. We see at least four ways that machine learning might facilitate knowledge acquisition for the knowledge requirements of expert systems (Shalin, Wisniewski, Levi, & Scott, 1987):

- *initial* construction of a knowledge base,
- refining *existing* knowledge bases,
- *adapting* an existing knowledge-based system, for example to accommodate user expertise or style, and
- providing a principled method for constructing knowledge bases of expert systems.

Thus, machine learning might replace portions of the ill-defined knowledge engineer-expert interaction. It provides a *formalized mechanism* for the knowledge acquisition process.

Under the AFWAL Crew Station Automation Technology (CSAT) program, we conducted a systematic study of learning-system technology with the objective of using this

technology in the development of pilot-aiding functions. We studied and thoroughly evaluated pilot-aiding functions, and defined the various approaches that have been used in the development of learning systems. We concluded that the highest-payoff pilot-aiding applications were the areas of tactical planning and information requirements. The most appropriate learning-system technologies for these applications were *deductive learning systems* such as EBL. The following conclusions supported our recommendation:

- EBL systems are the most capable of learning knowledge about procedures.
- EBL systems are preferable given an adequate domain theory.
- EBL systems make the best use of single positive learning experiences.

The Pilot-Aiding Problem

The defining characteristic of our problem domain is an expert system aiding a human performing a procedural task. DARPA and the Air Force's Pilot's Associate (PA) program is a large effort (greater than 100 man-years) to build such an expert system to aid aircraft pilots. Given the premise that the pilot is the final controller of the system and PA is an assistant, the two central modules of PA are the pilot vehicle interface and the tactical planner.

In a basic Pilot's Associate functional cycle, the tactical planner selects a plan to deal with particular circumstances. The tactical planner suggests this plan to the pilot via the pilot-vehicle interface. The pilot-vehicle interface judges the pilot's approval of the suggested plan, either based explicitly on the pilot's acknowledgement by voice or switch selection; or more likely, implicitly by successfully interpreting pilot actions in view of the plan. The pilot-vehicle interface is then capable of providing supportive information that allows the pilot to evaluate and execute the plan. In some cases the pilot may permit the pilot-vehicle interface itself to execute the plan. The pilot-vehicle interface also informs the tactical planner of pilot activity so that it may continue to support the pilot with options at each tactical decision point.

When the pilot-vehicle interface fails to interpret the pilot's actions in terms of plans suggested by the tactical planner, there may be several reasons. The pilot may be starting the first step of a new activity, the pilot may be in error, or the pilot may be executing an activity for which the Pilot's Associate lacks knowledge. In these cases, the system cannot anticipate pilot needs and support them with suggestions and information. The Pilot's

Associate will lag behind the pilot, redirecting its computational resources to the problem of getting back in step with the pilot, providing warnings if necessary, and maintaining general rather than specific support.

These interpretation failures due to lack of knowledge will constitute the initial learning experiences for two explanation based learning systems we are developing. The set of observed pilot actions and the goal they achieve will be input into one system and will allow us to add a new plan to the tactical planner knowledge base. In another system we will propagate the effect of learning a new tactical plan to the pilot vehicle interface, where we will use learning-system technology to add pilot information requirements for executing the new plan successfully.

Problems and Issues

Integrating our learning system with the PA performance system has raised a variety of theoretical and practical issues. There are issues associated with learning in a procedural domain. For example, generalizing over temporal relations is an open research question because of the propensity for interaction among the temporal relations of conjunctive subgoals. There are also challenges to using a planner in this domain. Because of the human in the loop, the planner must be able to generate plans that incorporate user actions that the system has observed. We refer to this as *partial plan completion*. The need for this capability makes the constraint posting aspect of our planning approach a necessary requirement. Other challenges for using a planner in this domain arise from the use of a macro library. A learning planner, by definition, must use a macro library. This raises all of the traditional schema activation and store versus compute tradeoff issues.

Another set of issues arises with respect to passing information between the performance system and the learning systems. The performance system generates the learning experiences for the learning system, but these experiences might require a significant amount of preprocessing before the learning system can use them. Conversely, the plans which the learning system produces will have to have certain types of information added before they can be inserted into the PA performance system. Finally, there are issues associated with creating the background information that the learning system must have before it can do any learning.

In the next section we describe the learning system we are developing and discuss the learning and planning issues in more depth. In the subsequent section we discuss the issues of passing information between learning and performance systems in more depth.

INTEGRATING A LEARNING SHELL WITH A PERFORMANCE SYSTEM

In this section we first describe our planning/learning shell. In our discussion of the planning/learning shell we first review the relevant planning and learning technologies, then discuss the issues that arise in combining these technologies for this problem domain, and finally we describe the basic modules in the shell.

Planning Background

The general AI planning problem is to search a set of pre-defined operators to find a sequence that will transform an initial-state description to a desired goal-state description (Pednault, 1985). The main focus of planning research for the last 10 years has been on conjunctive planning—the problem of achieving several goals simultaneously. The difficulty in conjunctive planning is in the interactions between the operators that achieve the individual goals. These interactions introduce additional constraints on the temporal orderings of the operators. Chapman (1985) has recently developed a formal analysis of the domain independent approaches to planning and has integrated these approaches into a general framework inherent in the control structure of his planning system, TWEAK.

TWEAK begins the planning process with a representation of the intial state and the conjunction of goals in the desired state. Working backwards from the desired state, it inserts operators into the plan until the operators link the initial state to the desired set of conjunctive goals. TWEAK generates plans that are non-linear, that is, partially ordered sequences of operators. This reduces the need to retract arbitrary orderings as interactions between operators are identified. To achieve a goal in a plan, the goal first has to be established (made true by an operator) and then declobbered (subjected to an analysis of its potential interactions with all of the operators that exist in the current plan). If declobbering evidences an interaction, a constraint on operator orderings or operator bindings is posted that eliminates the interaction. Given the persistence assumption (i.e., once a proposition has been asserted, it remains true until denied) the TWEAK planning algorithm has been mathematically proven to be "correct" and "complete". Because of this property, and because of it's built in non-linearity (discussed further below), we have selected the

TWEAK algorithm as the basis for generating explanations about sequences of actions in our Explanation-based Learning Shell.

Explanation -Based Learning Background

We are using DeJong and Mooney's (1986) EBL system for application to our learning problems. The main reason for selecting this approach was that the representation of EBL's knowledge base is compatible with those of the tactical planner and pilot-vehicle interface. In particular, we have built our system based on Mooney's (1987) Genesis system and Mooney and Bennett's (1986) Explanation Generalization Using Global Substitution algorithm (EGGS).

EBL learns schemata to improve its understanding of an agent's actions (e.g., the actions of a person or a robot). A schema is a partially ordered set of actions and/or simpler schemata that are causally related. That is, preceding actions or schemata provide the applicability conditions for succeeding actions and schemata.

Input to the system is typically a sequence of actions performed by some agent. The system attempts to understand these actions. By *understand* we mean that the system builds an internal representation in which all of the causal and motivational information is explicitly stated. Thus, the system must justify why each action was possible, how each state came about, and why each volitional action was desired by its agent.

Sometimes the agent's actions achieve a goal that is beyond the current understanding of the system. When the system recognizes that this has happened, it constructs an explanation of how the goal was achieved from its understanding of the agent's actions. An explanation is the portion of the causally complete representation concerned with achieving the goal in question. In other words, it is a data-dependency graph of causations showing the justification of why the goal is believed to have been achieved.

Next the explanation is generalized into a schema and stored away for future use. The generalization process consists of making a structural copy of the explanation with semantically empty nodes in place of the explanation's action, state, and object nodes. Then all of the constraints necessary to preserve the veracity of the explanation are imposed on this structure. This constrains the structure to have just those features required for the observed problem-solving technique to have been successful. Action nodes are then

resolved to the most general action known that fulfills the required constraints. The resulting generalized structure is filed away in the system's schema library for future problem solving and understanding. The acquisition of new schemata improves EBL's understanding of an agent's actions.

The way EBL represents its knowledge is similar to how knowledge is represented in the tactical planner and pilot-vehicle interface knowledge bases. Much of EBL's domain theory is represented as a hierarchy of schemata (the system also has inference rules in its domain theory). In the tactical planner and pilot-vehicle interface knowledge bases, knowledge is represented hierarchically with structures called *plan classes* and *scripts* that are similar to EBL's schemata. In addition, the problem of acquiring a new schema by understanding an agent's actions is similar to the LSPA problem of acquiring a new plan or script by understanding the pilot's actions.

Technical Issues in Combining TWEAK and EBL

The standard task of the explanation generator in EBL is to generate an explanation for why a learning experience is an example of some goal concept. This explanation is usually based on a logical deductive proof using the rules of inference of a domain theory. For procedural domains in which the rules of inference are based on actions executed over time, we argue that a simple logical proof is not sufficient. Explanation in procedural domains requires reasoning about the temporal relations between actions that enable, cause, or interfere with other actions. For this reason, we have substituted a planning algorithm designed for this kind of reasoning to generate explanations of tactical plans. We have selected a non-linear planner because of the similarity between the partially specified planning structure generated while planning and the partially specified plan represented by the sequence of actions executed by the pilot. Because of this similarity the planning system naturally incorporates the pilot sequence of actions as a constraint on the plan it generates to achieve the goal state.

There are a number of technical considerations that remain after a plan explanation has been generated. The additional temporal constraints in the plan make the generalization process more complex. Tradeoffs between over and under generalizing the plan to produce the correct macro need to be identified for the tactical planning domain. Overly general plans in procedural domains are costly when planned actions interfere with each other. On the other hand, generalized plans that are too specific will burden system performance time when

they place excessive demands on memory. Overwhelming the system with stored macros generated by learning is a pervasive concern. Initially the stored macros will improve performance time. However, eventually the addition of macros can overwhelm the system and performance time becomes worse.

Top Level Algorithm of the Learning Shell

The learning shell that we have built by combining EBL and TWEAK performs the following steps of explanation based learning:

- Generate a specific explanation of a partially specified plan
- Prune irrelevant details of the explanation
- Construct a global substitution list
- Package the resulting generalized explanation into a macro

We describe these steps in more detail below.

Generate a specific explanation of a partially specified plan—The TWEAK planning algorithm that we use begins with three inputs: 1) the initial state of the world before the plan was executed, 2) the sequence of recorded steps that the pilot executed and 3) the goal achieved by the sequence of actions. TWEAK outputs an explanation of the pilot steps in the form of an incomplete partially ordered plan that transforms the initial world-state into the achieved goal state. TWEAK achieves this plan by verifying the recorded steps and augmenting them with additional pieces of domain theory while maintaining the required temporal and logical relations.

Prune irrelevant details of the explanation—The next step in the learning cycle is to identify instances of existing macros that are used in the plan. That is, a completed plan will sometimes have a set of actions achieving a subgoal for which there is already an existing schema or macro. The purpose of this step is to replace these actions with the existing schema so that we do not needlessly duplicate existing macros.

Construct a global substitution list—After pruning, constraints that are necessary to retain the form of the explanation are collected into a global substitution list (Mooney & Bennett, 1986). This step determines which constants can be variablized and which variables must be the same.

Package the resulting generalized explanation into a macro—The plan is generalized by substituting the global substitution list into the plan. The generalized plan is then packaged into the required operator syntax and stored. The resulting macro can then be used by the explanation generator in constructing future explanations.

INTEGRATING LEARNED PLANS INTO THE PERFORMANCE SYSTEM

Basic Representation and Functional Cycle of a PA Tactical Planner and Intent Modeler

The Pilot's Associate program will supply learning experiences for LSPA. Interpretation failures in the basic functional cycle due to lack of knowledge will constitute the initial learning experiences for both the demonstrations. In the Learning System for Tactical Planning (LSTP) demonstration we will use this learning experience to add a new plan to the tactical planner knowledge base. In the Learning System for Information Requirements (LSIR) demonstrations we will propagate the effect of learning a new tactical plan to the pilot-vehicle interface, where we will use learning-system technology to add pilot information requirements for executing the new plan successfully. In this section we describe the basic knowledge representation and processes in the tactical planning and pilot vehicle interface systems that we will update with learning.

The tactical planner module itself consists of a hierarchy of planning elements arranged in a tree, with higher-level planning elements at the top, and decomposes into more specific subplanning problems at the tree's branches and leaves. Each planning element has the form of an object that can be activated by different messages from its parent planning elements. The activation messages are of two kinds: select and execute. Select knowledge determines how applicable this planning element is to solving the current problem. Each sibling planning element will calculate its own applicability metric for that subgoal. Plan elements continue to decompose themselves into subplanning elements by sending select requests. Once primitive actions are generated at the leaf nodes of the hierarchy, they are filtered up through their parent plan elements. During this process, plan integration, conflict resolution, and action combination take place to generate the completed tactic. Monitor rules determine when each of the actions should happen, and when a plan element is completed (transition) or failed. The execute message is the instruction to start or continue the execution of the planning element, instantiated with specific parameters that may or may not match the original selection parameters. When activated by an execute

request, a planning element will pass execute commands to the children plan elements best qualified to solve the problem.

The pilot-vehicle interface includes a pilot intent model and an interface manager. The pilot intent model receives pilot control inputs and their meaning, and classifies them as explained or unexplained. As the operator interacts with the system, he is free to pursue the higher-level known goals using whatever methods he chooses. The progressive identification of the operator's plans and goals results in an interactive update of known plans, goals, and scripts that are currently active. The interface manager ensures that the pilot receives all information necessary to perform the tasks assigned. To provide relevant information to the pilot, the interface manager selects a configuration of the interface that will present all important information and allow the pilot to control the aircraft in an effective manner, and sends configuration commands to display and control devices in the cockpit interface. The information requirements for each plan in Pilot's Associate are stored in individual plan elements. To identify the information requirements for a known plan class, the system simply locates the appropriate plan element in the pilot vehicle interface and sends its contents to the interface manager to be coordinated for display with information requirements from other concurrent plans, or other messages from the Pilot's Associate modules. Although the interface manager may request calculations to instantiate a plan element, the pilot-vehicle interface system has no internal capabilities for reasoning about the information requirements for a new plan. Thus, if the pilot-vehicle interface fails to identify the current pilot actions in terms of a known plan, it can only provide general information. If specialized information is required, the pilot must request it directly.

Integrating learned plans into the pilot-aiding system

There are three areas of concern for integrating the learned plans into the existing pilot-aiding performance system: a) the experiences used for learning must be relevant and useful to the performance system, b) an appropriate domain theory is required for explaining the learning experiences to support deductive learning, and c) the learned plans must be properly inserted into the knowledge base of the performance aiding system. Addressing learning in the context of a real-world performance system affects all of these concerns. We discuss them in turn below.

Obtaining, Selecting and Representing Learning Experiences— A judicious selection of learning experiences is needed to ensure that the learned information will be beneficial to

the functioning of the performance system. Learning systems can ensure the relevance of the added material by restricting their learning to failures of the performance system, when failure can be attributed to knowledge that is missing from the performance system. This approach has been adopted in learning apprentice systems (Mitchell et. al.) and systems that learn while problem solving (Laird et al., 19xx; Mitchell, 19xx). We adopt the same approach by selecting as learning experiences those cases in which the pilot-vehicle interface fails to recognize the behavior of the pilot. These learning experiences capture pilot behavior executed in performing the real task—a generally preferred approach to capturing human knowledge (Simon & Ericsson, 19xx).

Of course it is possible that the failure of the pilot-vehicle interface to recognize pilot behavior is due to a pilot error or irrelevant pilot behavior, and not a deficiency of the knowledge base of the performance system. A fully automated learning system would be able to discriminate between erroneous or irrelevant pilot behavior, and knowledge base deficiency. This is beyond state-of-the-art learning systems. We are currently hand-selecting the learning experiences from the set of pilot-vehicle interface failures and investigating the area of student-error modeling (Langley et al. 19xx; Sleeman, 19xx; Van Lehn 19xx) for an automated approach to this problem. An advantage of the user plan recognition domain is that the user can assist in screening out erroneous or irrelevant learning experiences.

A second issue in obtaining learning experiences in the context of a real performance system is representing them for an AI-based symbolic approach to learning. In the context of PA we have the option of characterizing pilot actions according to the command and control inputs fed into the pilot vehicle interface. In addition, the plane's sensors provide considerable information that characterizes the state of the world. Such a direct and thorough characterization of learning experiences is probably unusual and almost certainly helpful, but not without negative consequences. First, the information in its most primitive form is quantitative and must be interpreted symbolically in order to be used for explanation based learning. This is a familiar difficult problem. In AI it is referred to as qualitative reasoning (Forbus, 19xx); in military task domains it is referred to as sensor fusion. A second problem with the information associated with these learning experiences is that there is too much of it, and there are no clear a priori bases for deciding what should be preserved and what should be deleted in a proper (informative) representation of a learning experience.

Obtaining a Domain Theory—The domain theory required for deductive learning is at a level of analysis that is more fine-grained than the knowledge in the performance system. The required domain theory explains the knowledge represented in the performance system, but is not the same as the existing knowledge. Clancey (1984) makes a similar distinction in knowledge for performance and knowledge that explains performance in his system for explaining the process of medical diagnosis to interns. The domain theory that supports explanation based learning is used to explain the learning experiences and guide their generalization for eventual placement in the knowledge base of the performance system.

Thus, building an explanation based/deductive learning system requires a knowledge engineer to acquire and build a new, more detailed theory of the domain. This domain theory must be sufficient for constructing a proof of how the learning experience fulfills a goal concept. It must take as inputs the symbolic data structure that represents learning experiences and produce as outputs the data structure required by the performance system. Building this domain theory has all of the same difficulties inherent in building any domain theory--including the requirement to identify the appropriate primitives and the higher-order/abstract concepts used to control the search processes that operate on the domain theory. Furthermore, in real-world domains this theory may be incomplete, inconsistent or intractable—properties of domain theories that challenge state-of-the-art explanation based learning which is based on formal proofs. To make explanation based learning practical relative to hand-coded modifications, the considerable effort of constructing a suitable domain theory must be outweighed by the utility and breadth of the resulting learning system.

Inserting Learned Plans into the Performance System—Judicious selection of learning experiences and deductive learning based on a sound domain theory help to ensure that the learned material is appropriate for integration with the existing knowledge base of the performance system. It does not, however, solve all of the problems associated with this integration. We describe two sets of relevant issues: those that are local to the data structure itself and those that are global with respect to the entire performance system.

Local Issues Associated with Inserting Learned Plans: Learning by analyzing observed behavior may support recognition of that behavior if it occurs at a later point, or provide the knowledge for suggesting that plan in an intelligent aiding system. Recognition knowledge is probably not sufficient for executing that behavior, as any student of tennis or golf is

well aware. In state-of-the-art machine learning, we would have to add execution knowledge by hand in order to have a fully functional plan. For the more limited case of having a plan that can be used for plan recognition we may still have to add knowledge to the plan data structure that allows it to be indexed properly for the plan recognition system.

Global Issues Associated with Inserting Learned Plans: The learned plan has to be placed properly in the existing knowledge base. In the case of the knowledge base for the tactical planner for the PA system we are using, this means placing it at the proper level of abstraction in the knowledge base. It may also require repairs to both the learned plan and its pre-existing siblings so that all of the plans may be discriminated from one another for both plan recognition and plan suggestion functions. Finally, in the context of a complex suite of expert performance systems such as PA, adding knowledge to one of the performance systems creates new learning problems for the other performance systems. For example, adding a new tactical plan to the Tactical Planner creates a learning problem for the pilot-vehicle interface of defining the information for display that a pilot requires in order to execute that tactic.

CONCLUSION

Our project to use explanation based learning to acquire plans for the PA system is still in its early stages. It is already clear, however, that this project will force us to address many crucial issues that will often be overlooked in learning and knowledge acquisition research that does not work with a real performance system. By far the most challenging task for such learning systems will be constructing the background knowledge needed for the learning system. The learning domain theory is in addition to the performance system domain theory. The learning domain theory must be able to explain failures in the performance system's knowledge bases and perhaps also failures in the performance system's control structure. Thus, automating knowledge acquisition will always require a substantial effort, and developers must weigh the relative costs of hand coding and updating the performance system's knowledge bases versus the cost of building the learning system to automate these tasks.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

A CRITICAL LOOK AT KRITON

Moving from first to second generation

Marc Linster
Expert System Research Group
Institute for Applied Information Technology
GMD
PO Box 1240
St. Augustin
e-mail : linster@gmdxps.uucp

It is the purpose of this paper to develop a basis for second generation knowledge acquisition tools. We start out with a brief description of our current knowledge acquisition tool KRITON. We compare it to a set of other knowledge acquisition tools to highlight the different facets a future tool should cover.

We will review some of the criteria for second generation tools and state our point on the modeling of problem-solving processes.

Considering these criteria,i.e. the need for modeling, experience with KRITON and the comparison to other tools, we will sketch the features the successor system of KRITON will have.

Finally we will state our perspective on the long-term goals of knowledge acquisition systems.

1. INTRODUCTION

KRITON has been devised to be the knowledge acquisition front-end of BABYLON (Di Primio, Brewka 1985), a hybrid, general purpose knowledge-representation and inference tool.

To this purpose a set of weak knowledge acquisition techniques have been developed and integrated into an architecture allowing for incremental knowledge acquisition. The methods integrated are :

- PROTOCOL ANALYSIS (Ericsson, Simon 1984) for the acquisition of procedural and associative knowledge,
- INTERVIEW and TEXT ANALYSIS for the acquisition of static domain features.

They are called weak methods as no single one is strong enough to acquire knowledge for all domains and all problem-solving strategies. The methods have a strong psychological motivation and stress the elicitation facet of knowledge acquisition.

2. OUR ORIGINAL VIEW OF THE KNOWLEDGE ENGINEERING PROCESS

When we initially developed KRITON we adopted the "rapid-prototyping" view on knowledge engineering. We saw knowledge acquisition as consisting of two distinct phases :

- elicitation
- representation

This implies, that given a good inference and editing environment it is the task of the knowledge acquisition tool to provide as much "raw" knowledge as possible. As the operationalization of knowledge happens in the representation environment, KRITON did not have to incorporate application-dependant features. This allowed us to view KRITON as a problem- and domain-independent elicitation tool.

Background and introductory literature is analyzed to define basic concepts and their relations.

An INTERVIEW is conducted to gain more information on the semantic net and the concepts described therein.

PROTOCOL ANALYSIS provides information about how the concepts are used in the problem solving process, and what associations they trigger.

The final refine-debug cycle is done by the knowledge-engineer and happens entirely in the representation environment and its specialized editors.

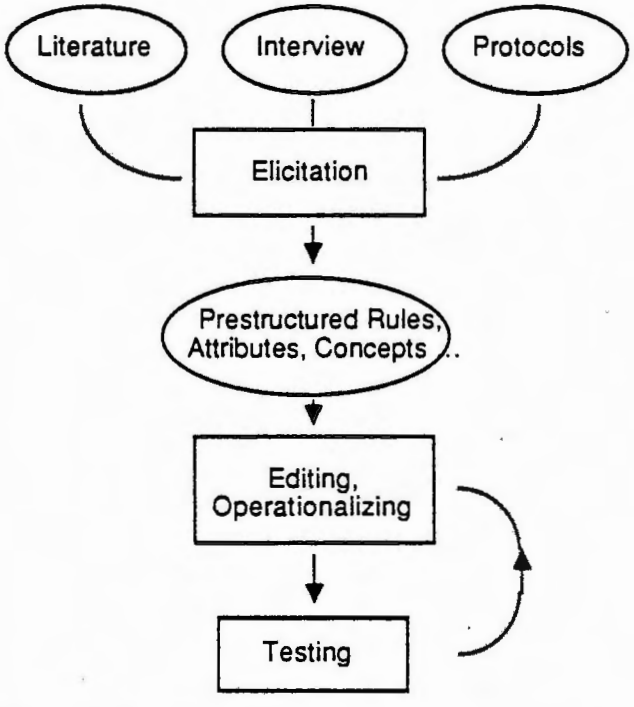


Figure 1: Rapid Prototyping View on Knowledge Engineering

Elicitation delivers rules, concepts and attributes describing these concepts. KRITON uses the expert´s terminology and structures the data the way the expert views his domain. This data is then transformed into an operational representation according to the needs of the interpreter. The representation environment provides all the operations and functions required once elicitation has been accomplished. The refine-debug cycle is thus limited to the representation and inference tool, except for new areas that are introduced into the knowledge base.

3. DESCRIPTION OF KRITON

In this section we give a concise description of the features of KRITON and its architecture. For a more in-depth presentation see (Diederich 1986, Diederich & Linster 1988).

Here we only want to provide a basic understanding of our arguments for a move towards a second generation of knowledge acquisition tools.

3.1. Text Analysis

Knowledge acquisition often starts out with the analysis of background literature. Thus the knowledge engineer gains a basic understanding of the domain elements and this allows him to ask meaningful questions during the first interviews.

Illustration :

An excerpt of an introductory text on programming languages is analyzed and transformed into one of many possible taxonomies representing aspects of the domain.[1]

*...**languages** that can be selected are : **LISP**, **COBOL**, **ADA**, **PASCAL**, **ASSEMBLER**, **C**, **C++** and **FORTRAN**. **FORTRAN**, **COBOL** and **ASSEMBLER** are **blockfree languages**, **LISP** is highly functional, **ADA** and **PASCAL** are procedural ...*

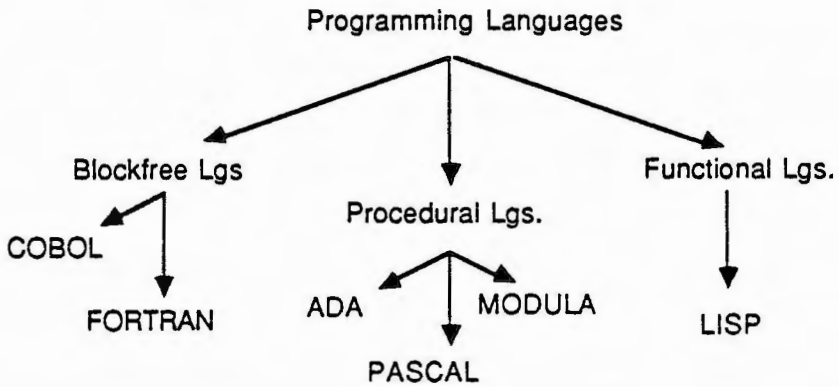


Figure 2: A Taxonomy Generated with TEXT ANALYSIS

This has motivated us to include a TEXT ANALYSIS feature in KRITON. Texts are read from a file, nouns are highlighted and made mouse-sensitive. This relies on dictionaries and on specific properties of German language. The knowledge engineer can include the nouns into a hierarchy (or heterarchy) describing the way the text presents the organization of the concepts of the domain, or the way the expert sees their organization. This does not have to be KL-ONE-like hierarchy relying on attribute-value refinement.

The TEXT ANALYSIS is a computer support to get started in a knowledge acquisition process.

3.2. Interview

The INTERVIEW component edits and completes the initial information gathered by the TEXT ANALYSIS. Relying heavily on Repertory Grid (Kelly 1955, Boose 1986) it acquires attributes describing the concepts of the domain. Questions are asked about differentiating attributes between concepts like :

[1]Nouns will be printed in bold here. In the KRITON system they are highlighted to indicate their mouse-sensitivity. We do not include screen copies as KRITON is limited to the analysis of German protocols.

What do MODULA and ADA have in common that differentiates them from PASCAL?

What differentiates PASCAL from ADA?

The answers to these questions are given by indicating an attribute that is applicable to all the concepts in the questions and that has differentiating attribute-values for the different concepts. If no sensible answer can be given the system changes into a laddering mode to explore possible hierarchical relations among the concepts not made explicit yet.

The concepts used for questioning are chosen from a common context[1]. This means that they have a common "super-concept" in the taxonomy. The relation the concepts have to the "super-concept" does not matter for this purpose, it does not even have to be specified. We assume that the expert grouped the concepts in the given manner because they have a common context.

Several other techniques are used in the INTERVIEW to complete the attribution of the concepts.

EXPLICIT INHERITANCE :

If the attribute *DATASTRUCTURE* is defined for BLOCKFREE_LGS the system will inquire about the corresponding attribute-value for COBOL and FORTRAN.

EXPLICIT GENERALIZATION :

The system will inquire about the value for the attribute *DATASTRUCTURE* and the concept PROGRAMMING_LANGUAGES. If this attribute not known, the user does not have to indicate it.

COMPLETION OPERATION :

Let´s assume it is known in the taxonomy that PROCEDURAL_LGS has *pointers*, *strings* and *chars* as *DATASTRUCTURES* and that none of ADA, PASCAL or MODULA has *strings* as a value of the attribute DATASTRUCTURE. The system will inquire about a concept comparable to ADA, PASCAL and MODULA that has *strings* as *DATASTRUCTURES*.

In KRITON these operations must be explicit as the user wants to edit and change the semantic net during the INTERVIEW. If inheritance, generalization or completion happened implicitly, nothing could be said about the values assumed if the edge that triggered the operation was erased or changed, e.g. from IS-A to SUBSET-OF. Making these operations explicit does not guarantee their truth or consistency either, but it makes sure that the expert agrees with them.

An ELICITATION COUNCELOR indicates the appropriate actions in the INTERVIEW component. To do so it compares the attribution of concepts closely related in the graph and proposes differentiating questions and completion techniques.

---

[1]The problem of context can not be neglected in repertory-grid questions, as many questions can only be given a sensible answer if there is a context that allows for such an answer. The question " What do diamonds and Persian carpets have in common that differentiates them from color-tvs?" can be answered if it is known that the context is household insurance-policies.

3.3. Protocol Analysis

PROTOCOL ANALYSIS is a technique that helps acquire procedural and associative knowledge. It is based on the work of Ericsson and Simon (Ericsson & Simon 1984) and concentrates on the analysis of transcripts of thinking-aloud recordings.

While the expert is solving a problem he speaks into a microphone and mentions his actions, conclusions and observations. The important feature of this kind of protocol is that the expert does not have the time to make his utterances sound reasonable, thus they can not be garbled due to inference processes not connected with the direct problem solving process. This is due to the fact that the expert verbalizes the information contained in his short-term-memory which triggers, in the problem-solving-process, the associations pointing into the long-term-memory.

To analyze the protocols, they are written down together with the pauses of speech. Based on the assumption that pauses of speech represent delimiters, comparable to punctuation marks, separating the transcript into coherent segments, these segments are transformed into operator-argument structures which are then combined into rules to make the inference steps, implicitly contained in the protocol, explicit.

PROTOCOL ANALYSIS is a partially automated and partially computer-supported feature of KRITON. Operator-argument structures are generated automatically. They can be post-edited by the user. He must then combine them into rules to make associative and procedural information contained in the protocol explicit.

Illustration

An excerpt of a thinking-aloud protocol is analyzed :

*...** large company** they do some research**got a good software department** what they want are COBOL code generators**they have used LISP before** let them use LISP**...*

It is transformed by the system into operator-argument structures, that have been post-edited by the expert :

(large D1 company[1]), (do D2 research), (software-department D3 good)
(need D4 Cobol-code-generators), (used-before D5 Lisp), (recommend D6 Lisp)

These are then combined into rules to make the inferences described in the protocol explicit :

```
If      (size-company D1 large)
        (do-research D2 t)
        (quality-software-dept D3 good)
        (used-before D5 Lisp)
        (task D4 Cobol-code-generators)
then    (recommend D6 Lisp)
```

[1]The markers point to the phrase in the protocol used to generate the operator-argument structure. These markers serve documentation purposes.

3.4. Interaction of the Techniques

TEXT ANALYSIS and INTERVIEW work on a common datastructure which is a semantic net consisting of labeled edges and attributes. The user has total freedom to define relationships as edges or as attributes. The PROTOCOL ANALYSIS produces rules describing how the concepts in the semantic net are used in a problem solving process. All the techniques compete in the completion of this internal representation, and as this procedure is purely incremental we do not see a need for conflict resolution at this point.

In a prototypical acquisition session we start out with analyzing a background text and building a taxonomy representing the most important and frequently used concepts of a domain. These will then be attributed and described in the INTERVIEW. The taxonomy can be edited to fit new facets of the domain unraveled during the INTERVIEW. After this we can go back to the TEXT ANALYSIS to complete the taxonomy by analyzing a second chapter of the background text, or directly go into the PROTOCOL ANALYSIS. TEXT ANALYSIS or INTERVIEW must be preliminary to PROTOCOL ANALYSIS as *"... preliminary to the encoding process, the task and the protocol are analyzed to extract the vocabulary of objects and relations needed to define the problem space and the operators"* (Ericsson & Simon 1984, p. 264). This allows us to refer to a well defined vocabulary and to defined concepts of the domain that are then used as a basis for the analysis of the operator-argument structures.

New concepts appearing in rules are introduced into the taxonomy in the INTERVIEW.

The result of an acquisition process with KRITON is a dense network of relations between concepts describing structural relations (stemming from INTERVIEW and TEXT ANALYSIS) or associative relations (stemming from PROTOCOL ANALYSIS).

4. COMPARISON TO OTHER TOOLS

We will now compare KRITON to a selection of other knowledge acquisition tools:

- ETS / AQUINAS
- KSS0 / KITTEN
- OPAL / PROTEGE
- SALT
- KPT
- ROGET

This is not an extensive selection. In this list each tool stands for a particular view on knowledge acquisition : ETS/AQUINAS are elicitation tools geared towards the generation of operational knowledge bases, KSS0 / KITTEN are analysis tools, OPAL/PROTEGE are specialized knowledge editors, KPT is a modeling support tool. We listed ROGET as it is a first attempt to capture knowledge about the conceptual problem-solving model.

All of these are important aspects for second generation knowledge acquisition tools.

4.1. ETS / AQUINAS

4.1.1. Description. ETS (Expertise Transfer System) (Boose 1986) is based on Repertory Grid Techniques. It uses this information to generate implication rules among traits and classification rules for diagnostic systems. It makes use of Hinkle's (Hinkle 1965, in Boose 1986) WHY and HOW questions to broaden its information and to introduce hierarchical structures. AQUINAS is an extension of ETS that allows elicitation from multiple experts, decomposition of problems and incremental testing. Both ETS and AQUINAS generate operational knowledge bases. They have an internal, implicit model of classification problem solving and are not suitable for synthesis application (Boose 1985).

4.1.2. Comparison. KRITON integrates more acquisition techniques than ETS or AQUINAS do, but it does not make such extensive use of their features. It does not include an internal problem solving model either. This means that ETS and AQUINAS are stronger tools for those applications that they have been tailored for and the expert needs less guidance by a knowledge engineer to build a knowledge base. But outside of these problem domains, when an expert system is to be build, and it is not known what problem solving strategy is to be used, KRITON is certainly competitive through its larger spectrum of elicitation techniques.

4.2. KSS0 / KITTEN

4.2.1. Description. KSS0 (Knowledge Support System Zero)(Gaines &Vickers 1987) and KITTEN (Knowledge Initiation and Transfer Tools for Experts and Novices) (Shaw & Gaines 1987) are two tools based on Personal Construct Theory using Repertory Grids as main elicitation techniques. They both analyze the grids using ENTAIL, SOCIO, PRINCOM and FOCUS. ENTAIL (Gaines & Shaw 1981) is an algorithm showing implications and subsumptions between traits and elements. FOCUS and PRINCOM are clustering procedures, SOCIO is an algorithm comparing repertory grids from several experts to decide upon their agreement, understanding and consistency. KSS0 generates operational knowledge bases for NEXPERT OBJECT.

4.2.2. Comparison. KSS0 and KITTEN have an implicit classification problem solving model as ETS and AQUINAS do. This restricts them to certain problem types. But the analysis features included seem so strong that KSS0 and KITTEN can be used independently from the construction of operational knowledge bases just to analyze the experts knowledge and expertise. This is certainly an important trait that KRITON does not match.

4.3. OPAL / PROTEGE

4.3.1. Description. OPAL(Musen et al. 1986), is the knowledge editor for a cancer therapy management system called ONCOCYN. It does not only have an internal model of the problem solving process (skeletal plan refinement) but as it is domain specific (prescription of cancer-treatment protocols) it has a task model. This way it can ask questions about symptoms, cures, treatments, protocols and so on. It uses forms that the expert can fill out, from which operational knowledge bases are generated.

PROTEGE (Musen 1988) is a tool that creates graphical knowledge-editing tools for problem-solving methods that can be described as plan-refinement.

The important new feature of PROTEGE is that the problem solving model, even though encoded into the system, is made explicit.

4.3.2. Comparison. OPAL and PROTEGE use forms to acquire knowledge. That means that their use is only possible if the expert can easily express his knowledge. In this sense they both aren't elicitation tools. Their task- and domain-models are their strong points.

KRITON is not directly comparable to either, as it could serve as a front-end to OPAL to define vocabulary, strategy, examples of scripts or treatment-protocols.

4.4. SALT

4.4.1. Description. SALT (Marcus 1986) is a knowledge acquisition tool designed for synthesis problems . It has been used to configurate elevators in the system VT (Marcus et al. 1988). It supports a backtracking strategy. To do so knowledge about procedures used to define or compute certain values and about constraints relating these values is acquired. In the configuration system these constraints will enforce backtracking if violated.

It is SALT's task to provide the building blocks for the plans that the underlying expert system uses to generate the construction specification.

4.4.2. Comparison. Quite like OPAL, SALT cannot be considered an elicitation tool and is therefore not directly comparable to KRITON. It is more of a very specific, and therefor very smart editor for a special kind of knowledge as it has a narrow problem solving model.

This strong use of the model makes it very interesting. KRITON could again be considered a front end for SALT in domains where vocabulary and problem stages are not yet well known.

4.5. KPT

4.5.1. Description. The KPT (KADS Power Tools) (Anjewierden 1987) provide the bottom-up support for knowledge acquisition with the KADS methodology (Breuker et al. 1987). KPT include a concept-editor, a hyper-text like protocol-editor and the KADS-interpretation-model librarian helping to choose an interpretation model from a library.

4.5.2. Comparison. Looking at the KADS-POWER-TOOLS we find strong editing, managing and design support to devise KADS-interpretation models. Hyper-text-based text- and protocol-analysis provide elicitation-support for the knowledge-engineer. The protocol-analysis facilities are totally different from those of KRITON. They do not present any automated phases, but are limited to tools allowing the knowledge engineer to impose a structure upon the text. This implies however that more kinds of protocols and texts can be analyzed. KRITON is currently limited to analyzing concurrent thinking-aloud protocols.

4.6. ROGET

4.6.1. Description. ROGET (Bennett 1985) is a knowledge based system that helps the knowledge engineer with the acquisition of the conceptual structure described in a set of domain-independent terms, e.g. advice or result of the system, the evidence or data available to the system and the possible inference steps. ROGET contains a set of diagnostic tasks an expert system might perform, e.g. determine-problems, determine-causes, recommend-actions, predict-observations. One very interesting point is that ROGET knows about the problem-solving strategies of well-known expert systems. The user can then, by analogy select a fitting problem-solving method.

4.6.2. Comparison. ROGET is, unlike KRITON, not concerned with the elicitation of symbol-level knowledge, but it aims at the conceptualization of the procedures to solve problems. These will then in turn use symbol-level knowledge. It seems to us that ROGET is an important step towards domain-independent acquisition systems. The ideas presented by Bennett strongly influenced the conception of model-based knowledge acquisition.

5. SOME REQUIREMENTS FOR SECOND GENERATION TOOLS

Literature shows a rich variety of knowledge acquisition tools ranging from elicitation-oriented tools (KSS0, KRITON, ETS, AQUINAS, MACAO(Aussenac et al. 1988)...), special-purpose knowledge-editing front-ends (SALT, OPAL,...), modeling-support-systems (KADS-Power-Tools) to knowledge-editing-environments (KREME(Abrett & Burstein 1987) ...), not to forget learning systems and modeling methodologies (KADS...). Most of the techniques presented seem evolved enough to be integrated in a more comprehensive system. It would not be limited to one domain, one task, one problem-solving strategy or to supporting one phase of the knowledge acquisition process.

5.1. Requirements for Knowledge Acquisition Systems

Brian Gaines (Gaines 1988) lists the qualities a second generation knowledge acquisition system should have :

> •*Integrate a wide variety of weak methods encompassing the diversity of forms and sources of knowledge;*
>
> • *Integrate knowledge acquisition with system operation to facilitate a heavy activity of continuing system maintenance and upgrade as knowledge changes;*
>
> ...

Mildred Shaw (Shaw, Gaines 1987) lists requirements for a knowledge support system (KSS) :

> *The KSS tools should be domain independent;*
>
> *The KSS tools should be directly applicable by experts without intermediaries;*
>
> *The KSS system should be able to encompass a diversity of forms of knowledge and relationships between knowledge;*
>
> *The KSS system should be able to present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations;*
>
> *As much of the operation of the KSS should be funded on well-developed and explicit theories of knowledge acquisition, elicitation and representation;*
>
> ...

Both Shaw's requirements and Gaines qualities do not explicitly mention the need for higher level representation mechanisms to formulate problem-solving behavior.

We think though that the specification of higher levels, not only of abstraction and control, but levels of documentation and mediation is absolutely necessary. "Another im-

portant case of mixed models ... is the use of the knowledge level to characterize components in a symbol-level description of a system" (Newell 1982, p. 115).

5.2. A case for modeling

A simple example shall illustrate the need for higher level descriptions to document how knowledge at the lower, or symbolic level is to be used to solve a problem.

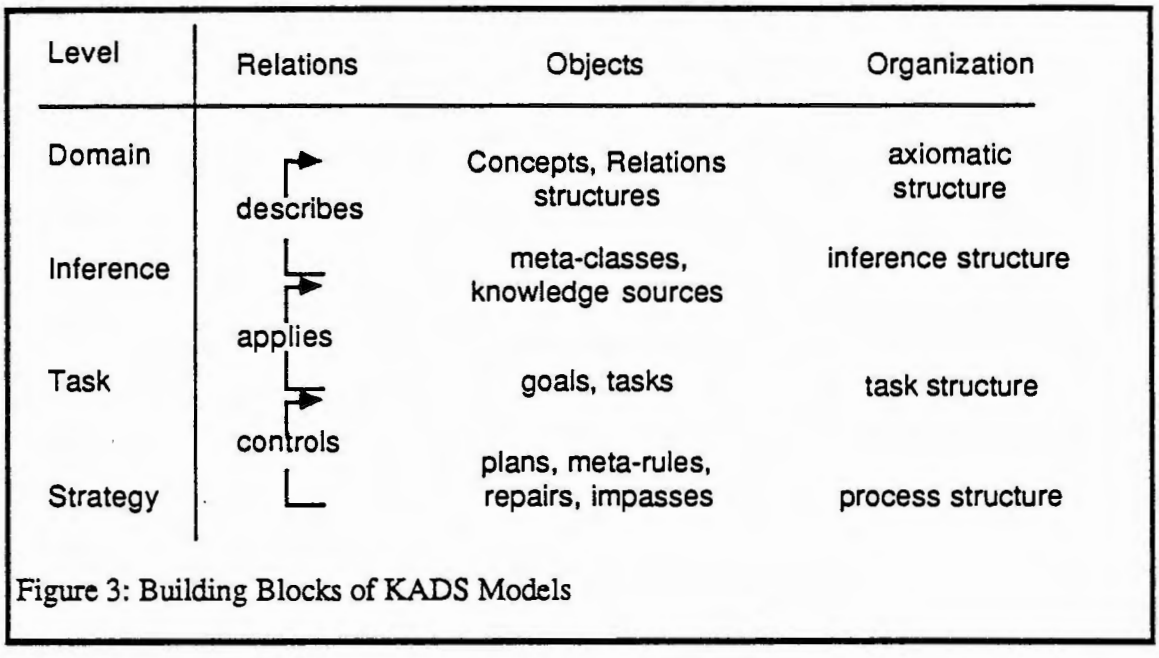


Figure 3: Building Blocks of KADS Models

Consider a mushroom classification system. The mushrooms are unequivocally described by a list of properties. The classifier uses heuristic match.

Toadstool	*Champignon*
...	...
...	...
taste : (bitter, sickening)	*taste : sweet*
...	...

If at some moment in the problem solving process toadstool and champignon were the sole possible solutions, the classification system could ask for the attribute TASTE, which is unreasonable to do if poisonous or sickening mushrooms have not been excluded with great certainty from the set of potential solutions.

This should make clear that even for simple systems it is necessary to make the problem solving model, i.e. the knowledge about how to use symbolic knowledge to solve a problem, explicit. The representation we favor is the one chosen in the Knowledge Acquisition and Design Structuring methodology (KADS).

Another most interesting aspects of KADS is its capability to represent design decisions of the knowledge based system in the system, i.e. represent the arguments that lead to a specific task structure in the strategic level, thus giving the system the facility to build alternate task structures if the strategy changes.

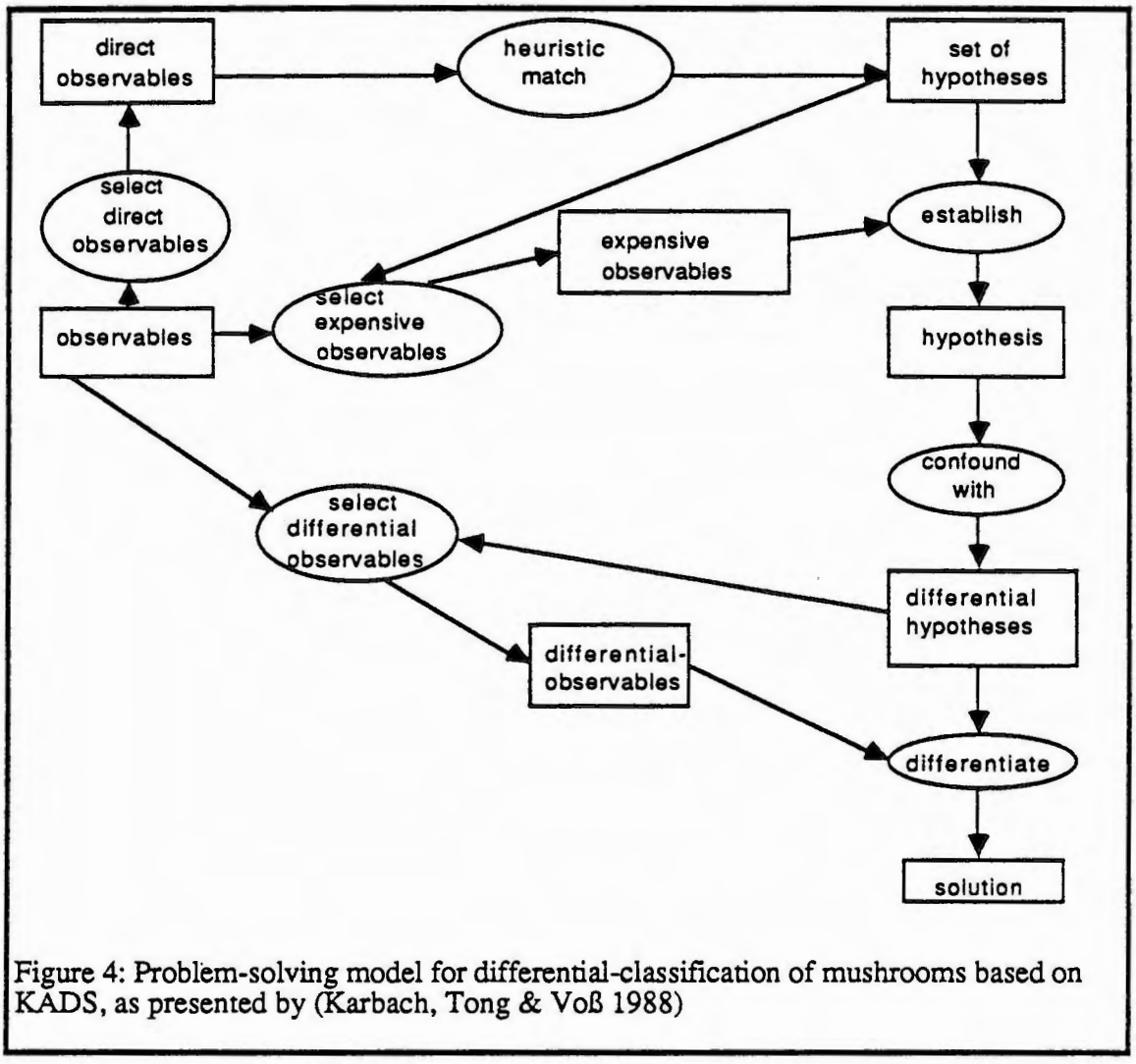


Figure 4: Problem-solving model for differential-classification of mushrooms based on KADS, as presented by (Karbach, Tong & Voß 1988)

That way systems "know" about their strategies and might change them. These changes may not only affect the task level, but may very well propagate through all levels. To us this reflective capability is a precondition for systems that perform autonomous knowledge acquisition. We are aware of the fact that this will not help us overcome the fundamental blindness of symbolic systems : *"In order to produce a set of rules ... it is first necessary to preselect the relevant factors and thereby cut out the role of the background. ... this process, by its very nature creates blindness."* (Winograd & Flores 1985, p. 133).

6. OUR REVISED VIEW ON KNOWLEDGE ENGINEERING

Our original view on knowledge engineering, relying on a separate elicitation and representation phase and neglecting the modeling phase proved inadequate. If we want to give extensive computer support to all phases of knowledge engineering we must :

- provide strong elicitation methods accessing a diversity of knowledge sources, and making the tool directly applicable by experts without intermediaries;
- provide means to model problem-solving processes on a higher level than the symbol level, i.e. we must access the knowledge level. This is important to make the tool domain- and task-independent;

- allow for editing of the internal representation.

We will call this model-guided knowledge acquisition. Several new problems arise :

- given a modeling language (KADS or GENERIC TASK (Bylander, Chandrasekaran 1986)) and data from the elicitation phase, how can we devise a model that describes the experts problem-solving behavior accurately ?

- if we move from our internal representation to an operational representation in a conventional expert system tool, how can we move back into the elicitation phase ? This is problematic as the external representations used in the elicitation are not isomorph to the internal representation used for modeling and inferencing, e.g. a set of rules can be deduced from many different repertory-grids using analysis methods like ENTAIL.

- the internal representation must be editable by the knowledge engineer. This implies however that the relation to the original data, e.g. protocols or transcripts of structured interviews might be lost. Even if the information contained were still the same, it would not be the experts words anymore. This will be problematic in subsequent elicitation phases where a knowledge base is incremented or changed due to unsatisfactory test results.

Elicitation accesses a diversity of knowledge sources.

The knowledge acquired is represented in semantic network like form.

It is analyzed to find an adequate model that explains the use of the knowledge in the problem-solving process.

This model is used to structure the representation. This structure is filled in (Musen 1988b) using elicitation methods.

The structured representation is operational in the sense of some conventional problem solver.

As shown in figure 5, the modeling view on knowledge engineering is an extension of the rapid prototyping view sketched in figure 1. The same extensions, i.e. modeling and structured representation will be applied to the original KRITON to create its successor system.

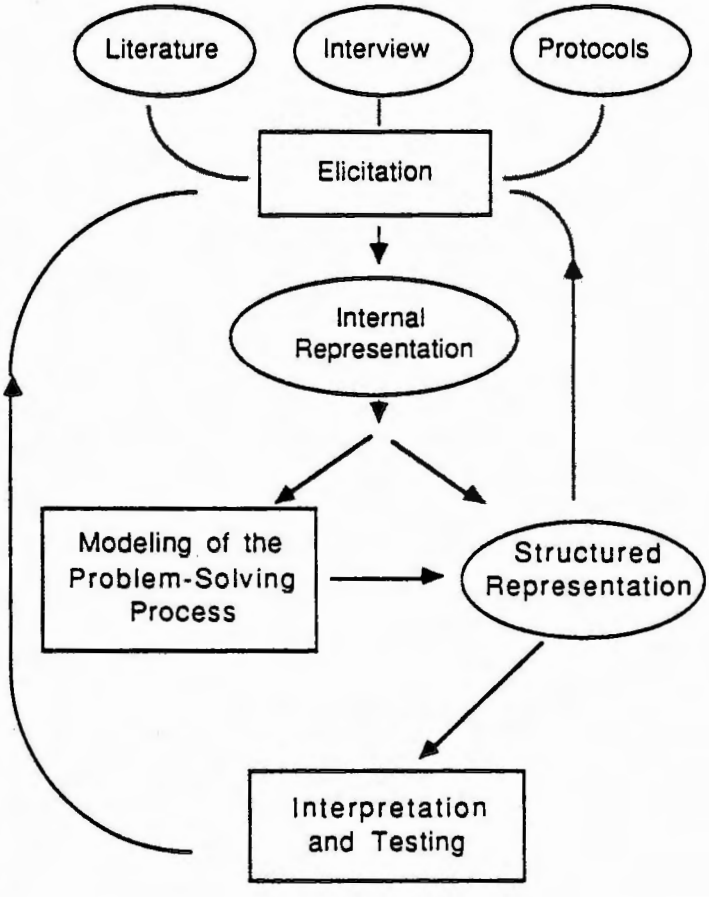


Figure 5: Modeling View on Knowledge Engineering

7. WHAT THIS MEANS FOR SOK

In this section we want to give a first description of the successor of KRITON (SOK) whose architecture will obey the paradigm sketched above.

7.1 Internal Representation

The internal representation must be very flexible to allow great freedom of expression. Ideally the internal representation will combine features of inference networks and hypertext (media) systems. Hypertext will provide the necessary unconstrained means to represent knowledge the way the expert formulates it. We may then impose a step-wise refinement to move from human interpretation to machine interpretability, i.e. from free-text formulations to operational inference networks. No information may be lost on this path, as the inference net will be imbedded in the hyper-text graph, being the machine-interpretable subgraph. That way design decisions, underlying assumptions, protocols and grids used to generate the operational features can be kept as a part of the knowledge base.

7.2 Elicitation Techniques

Techniques must be provided both to acquire tacit knowledge, i.e. the one really needing elicitation and knowledge that is more overt and that can easily be captured using forms or direct encoding into the internal representation.

For tacit knowledge we will integrate repertory-grid-like techniques, protocol analysis, clustering methods and procedures to analyze descriptions of concepts comparable to ENTAIL.

Give a differential description of CAR and BUS :

CAR	*BUS*
has 4 wheels	*has 4 - 6 wheels*
seats 4 people	*seats many people*
need a regular license	*got to have a special license*

In a next step this is transformed in to attribute-value structures :

CAR	*BUS*
nbr-wheels (4)	*nbr-wheels (4, 6)*
nbr-passengers(4)	*nbr-passengers(< 60)*
driving-permit (class-II)	*driving-permit (class-IV)*

These attribute-value representations will then be used for implication analysis.

Test for implication between :
left side : nbr-wheels, nbr-passengers and right side : driving-permit

nbr-wheels(4) and nbr-passengers(4) → driving-permit(class-II)

This implication may then be broken through the introduction of the concept TAXI.

We insist on transforming the descriptors into attribute-value pairs after a larger set of concepts have been described as then the expert has a better understanding of his descriptors. In quite a few experiments we noticed that experts made conflicting use of descriptors in the initial phases of an elicitation process, and that a second phase is necessary were they can say "well this is what I really meant", and reorganize their descriptors and group them around a smaller set of attributes.

As mentioned in the description of KRITON, context is a non-trivial question if repertory-grid is to be used. We think that the INTERVIEW described above sticks with the theory of personal constructs, but that the freedom of expression is much greater. Even if the given context does not allow for a meaningful differentiation between the concepts the user of the system can still give reasonable descriptions of the concepts. That way we can use his answers to define contexts, i.e. concepts belong to a common context if they share attributes.

This in turn allows us to switch from taxonomies that the expert generated to contexts in which we ask differentiating questions. This is done in KRITON now. In this new INTERVIEW we have attributes that can be used to generate taxonomies. One important thing is that we must allow for several taxonomies on one set of concepts, and that taxonomies may not be reduced to KL-ONE like hierarchies. We must allow for free formulation of categories independently from equality or refinement of attribute-values, e.g. in a taxonomy a stool must be discernible from a table, even though its attribute-values are refinement of the chair´s, i.e. height-of-table (15 cm - 100 cm), height-of-stool (20 cm - 60 cm) and top-surface-of-table (0.5.m^2 - 2 m^2), top-surface-of-stool (0.5 m^2 - 0.8 m^2).

7.3 The Modeling Issue

Our modeling language will be in accordance with what has been proposed for KADS. The underlying domain layer will be the symbol-level representations of inference-rules and facts. Metaclasses and knowledge-sources will, like a level of abstraction, impose structure on the lower level. The task level will describe how knowledge-sources are combined to form a problem-solving process. The highest level of KADS, the strategic level will only be present in the form of free-text documentation of design decisions.

Illustration :

Domain Level :

If weight (< 5 tons) and nbr-passengers (< 10) then class (small)

Inference Level :

Meta-classes : numeric-descriptors , qualitative-descriptors
Knowledge-source : Feature-abstraction-of-busses

Task Level :

To find a suitable driver

Feature-abstraction-of-busses
Feature-abstraction-of-drivers
Match features

7.4. How to Combine Elicitation and Modeling

We specify our problem-solving processes using knowledge-sources and meta-classes. Two questions arise :

- how do we find out about the knowledge sources and the meta-classes, i.e. which ones do we have, what are their characteristics and how do they fit into a task-structure?
- how do we use the knowledge sources and meta classes to guide elicitation ?

For the first question we have to rely totally onto the knowledge engineer's intuition, as no procedure is known to us that would be able to explain "what the expert really does" in unexplored domains, and those are the domains that knowledge engineering is all about. Heuristic match of known task-structures upon new problems as described by Bennett in ROGET might be a possibility.

For the second point two answers are possible :

- if we view knowledge sources as relations among meta-classes, and we know one instance (on the domain layer) of the corresponding meta-class, and we have a suitable taxonomy of the domain layer concepts, then we can try to fill in the relations through repertory-grids, forms and structured interviews.
- Eugene Charniak proposes (Charniak 1988) a story comprehension scheme that matches possible motivations against real stories to understand the reasons (or motivations) behind actions described in the stories. This is very similar to finding applicable task structures for protocols of the experts activities. The vast world and domain knowledge needed a priori for this matching procedure is a serious problem, as it really requires in-depth modeling of the domain before knowledge acquisition starts.

In SOK we will opt for the first scheme. An example will illustrate this.

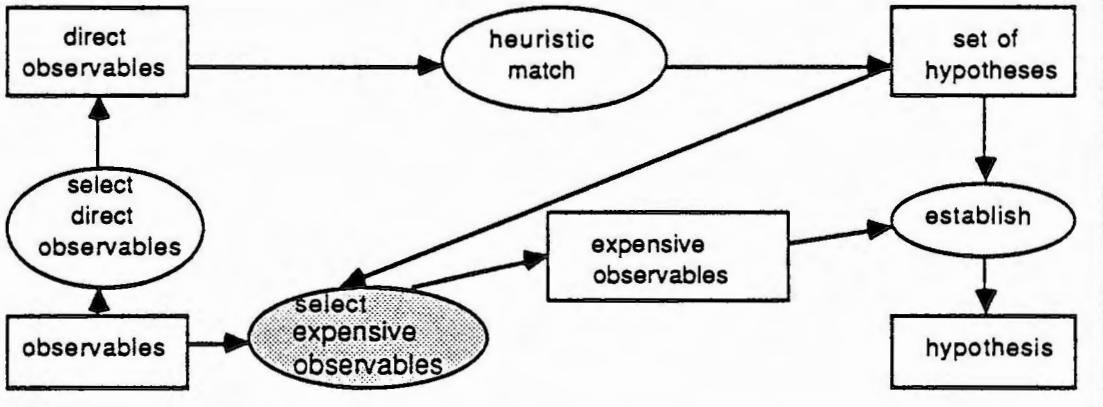


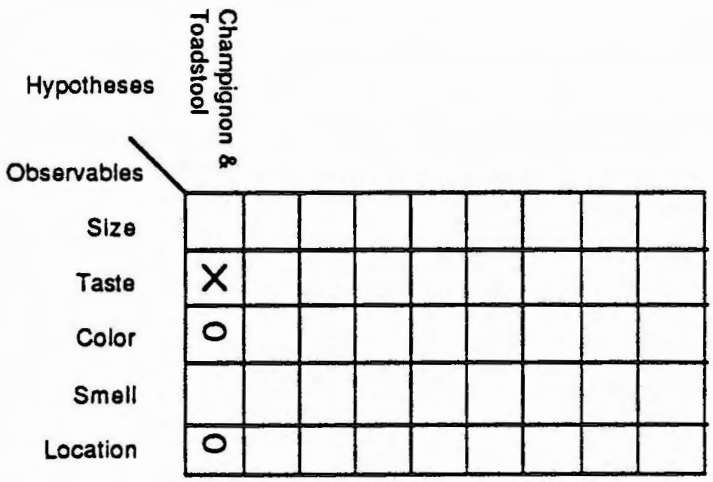
Figure 6: An excerpt of the task structure that was presented in figure 4.

The observables are variables describing the mushrooms, e.g. *season, location, size, smell, taste, color*. The hypotheses are subsets of the sets of all known mushrooms. We will try to elicit the knowledge source *SELECT_EXPENSIVE_OBSERVABLES*. We know that from the set of OBSERVABLES it selects a subset of EXPENSIVE_OBSERVABLES, and that the criteria for this choice come from the metaclass

SET_OF_HYPOTHESES. It is the purpose of this knowledge source to reduce the number of solutions, after a first application of initial criteria. Such initial criteria could be: *season, continent, altitude*. The additional criteria are called EXPENSIVE_OBSERVABLES to indicate that there is some metric that indicates that they should be the second choice; e.g. *season* is a "cheaper" attribute than *taste*, as in general finding out about the season is less dangerous than biting into an unknown mushroom.

We devise a grid :

This grid has as concepts the elements belonging to the meta-class OBSERVABLES. The expert is asked to differentiate among the OBSERVABLES, to tell us under which condition an observable is to be included into the meta-class called EXPEN-SIVE_OBSERVABLES. The constructs that the expert uses to do this should belong to the meta-class SET_OF-_HYPOTHESES.



In our example this means : *color* and *location* are useful attributes when toadstool and champignon belong to the set of hypotheses, *taste* must not be an EXPENSIVE_OBSERVABLE in this case.

This technique is strongly based on the idea of context (see footnote in section 3.2). In the above question the context is the selection of expensive observables.

The question could be formulated as follows : " Give a set of mushrooms for which color and location are useful observables, and for which taste is not applicable."

This questioning technique is influenced by repertory-grid, even though at first sight it may look totally unfamiliar. What we try to do is find the personal constructs that the expert uses to differentiate among concepts. The only new thing is that we make the context of the question explicit. Looking at original grids made by Kelly (in Sader 1980 p.141) we see questions like : "What do mother and father have in common that differentiates them from brother?", or "What do a successful person and a happy person have in common that differentiates them from an ethical person ?". In all these questions the context is implicitly contained as *minimal context.*

We are aware of the limitations of the procedure proposed. Extensions of relations can not be acquired this way. Explanation based learning could help increase the efficiency.

Another limitation is due to the two dimensions of the grid. To inquire about more complex relations and associations we may have to use more complex forms.

7.5. Internal Representation of the Modeling Constructs

The above specified view on knowledge sources implies that in the internal representation the building blocks of tasks, i.e. the knowledge sources, will be formulated as relations,

by grouping together inferences on the domain layer. Thus, on the internal representation, the structuring imposed through the modeling process will be like a second layer on the inference net. It is expressed in terms of the graph, but is not responsible for reasoning or inference procedures. The fact that they are expressed in terms of the internal representation makes them accessible for meta-problem solvers.

In later stages of the development of a knowledge-based system using SOK these "teams" of inferences representing one knowledge source may be used to define rule-packages responsible for certain subtasks of the overall problem solver.

7.6 Things not Modeled Yet

Our description of a potential knowledge acquisition tool has purposely left some issues out :

- Modality, i.e. describing how the system will be used, e.g. tutoring, decision support or sharing tasks with the user,
- Requirements, i.e. efficiency, interface, communication with devices etc.

To us it is an open question if, or to what extent systems can be developed without considering these two features, especially as modality may already be of high importance when the system is tested.

8. PERSPECTIVES

SOK will not be our ultimate in knowledge acquisition systems. We are fully aware of the need for systems to be "out in the world" to become fully intelligent and knowledgeable. Some design features of SOK show our goal to devise systems with architectures that make competence assessment possible through reflective features, thus allow the system to know about itself, and ultimately to modify itself if the tasks (the environment) requires this.

Acknowledgements

Thanks to Werner Karbach, Thomas Christaller and Barbara Becker for the discussions on the modeling features presented here, to Hans-Werner Güsgen and Karin Lagrange for making the paper readable.

References

Abbrett, G., Burstein, M.H., The KREME Knowledge Editing Environment, International Journal of Man-Machine Studies, 1987 (27), p. 103-126

Anjewierden, A., The KADS System, in : Addis, T.R., Boose, J.H., Gaines, B.R. (eds), Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems, Reading, U.K., 1987

Aussenac,N., Frontin, J., Soubie J.-L., MACAO: A Knowledge Acquisition Tool for Expertise Transfer, in : Boose, J.H., Gaines, B.R., Linster, M. (eds), Proceedings of the European Knowledge Acquisition Workshop 88, GMD, St. Augustin 1988

Bennett, J.S., ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. Journal of Automated Reasoning 1985 (1), p. 49-74

Boose, J. H., A Knowledge Acquisition program for Expert Systems based on Personal Construct Psychology, International Journal for Man-Machine Studies, 1985 (23), p. 522

Boose, J. H., Expertise Transfer for Expert System Design, Elsevier, Amsterdam 1986

Breuker, J., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Bredeweg, B., Wielemaker, J., Billaut, J.-P. Davoodi, M., Hayward, S., Model Driven Knowledge Acquisition, Memo 87, Esprit Project 1098, Amsterdam 1987

Bylander, T., Chandrasekaran, B., Generic Tasks for for Knowledge Based Reasoning:The right Level of Abstraction for Knowledge Acquisition, in : Boose, J.H., Gaines, B.R. (eds.), Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986

Charniak, E., Motivation Analysis, Abductive Unification, and Nonmonotonic Equality, Artificial Intelligence 1988 (34) No. 3

Diederich, J., KRITON: a knowledge acquisition tool for expert systems, in : Boose, J.H., Gaines, B.R. (eds.), Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986

Diederich, J., Linster, M. , Knowledge Based Knowledge Elicitation, in : Guida, G., Tasso, C.(eds.), Topics in Expert System Design, North-Holland, Amsterdam, in print

Di Primio, F., Brewka, G., BABYLON: Kernel System of an integrated environment for expert system development and operation, in : Proceedings of the 5th International Workshop Expert Systems and Applications, 573-583, Avignon, 1985

Ericsson, K. A., Simon, H. A., Protocol Analysis : Verbal Reports as Data, MIT Press, Cambridge, 1984

Gaines, B.R., Shaw, M.L.G., New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in : Shaw, M.L.G. (ed.), Recent Advances in Personal Construct Technology, Academic Press, London 1981

Gaines, B. R., Vickers J., Instantiation of Knowledge Structures in Books and Hypermedia, Technical Report KSI/NML-87-BHM, University of Calgary, Calgary 1987

Gaines, B.R., Second Generation Knowledge Acquisition Systems, in : Boose, J.H., Gaines, B.R., Linster, M. (eds), Proceedings of the European Knowledge Acquisition Workshop 88, GMD, St. Augustin 1988

Hinkle, D. N., The Change of Personal Constructs from the Viewpoint of a Theory of Implications, Ph. D. Dissertation, Ohio State University, 1965

Karbach, W., Tong, X., Voß, A., Filling in the Knowledge Acquisition Gap : via KADS´ Models of Expertise to ZDEST-2´s Expert Systems, in : Boose, J.H., Gaines, B.R., Linster, M. (eds), Proceedings of the European Knowledge Acquisition Workshop 88, GMD, St. Augustin 1988

Kelly, G.A., The Psychology of Personal Constructs, Norton, New York 1955

Marcus, S., Taking Backtracking with a Grain of Salt, in : Boose, J.H., Gaines, B.R. (eds.), Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986

Marcus, S., Stout, J., McDermott, J., VT : An Expert Elevator Designer that Uses Knowledge-Based Backtracking, AI Magazine, 1988 (9) No. 1

Musen, M.A., Fagan, L.M., Combs, D.M., Shortcliffe, E.H., Using a Domain Model to Drive an Interactive Knowledge Editing Tool, in : Boose, J.H., Gaines, B.R. (eds.), Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, 1986

Musen, M.A., Generation of Model-Based Knowledge Acquisition Tools For Clinical-Trial Advice Systems, Ph. D. Dissertation, Stanford University, 1988

Musen, M.A., Conceptual Concepts of Interactive Knowledge Acquisition Tools, in : Boose, J.H., Gaines, B.R., Linster, M. (eds), Proceedings of the European Knowledge Acquisition Workshop 88, GMD, St. Augustin 1988

Newell, A., The Knowledge Level, Artificial Intelligence 1982 (2), p. 87- 127

Sader, M., Psychologie der Persönlichkeit, Juventa Verlag, München 1980

Shaw, M.L.G., Gaines, B. R., , KITTEN: Knowledge initiation and transfer tools for experts and novices, International Journal for Man-Machine Studies (1987) 27

Winograd, T., Flores, F., Understanding Computers and Cognition : A New Foundation for Design, Ablex Publishing Co., New Jersey 1985

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Knowledge Acquisition For Planners *

Dirk E. Mahling
W. Bruce Croft
Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

September 10, 1988

Abstract

Previous work in knowledge acquisition has primarily addressed rule based expert systems and concept hierarchies in knowledge bases. The special constraints in the acquisition of knowledge for planners have not been explicitly identified. In this paper, we identify the requirements that situation calculus based planners pose on the knowledge acquisition process and compare these requirements to the techniques currently used or proposed for rule based expert systems. We then propose a framework for the human representation of subject performed tasks. To verify the validity of the framework, we conducted several experiments with more than 150 subjects. The last part of this paper describes the DACRON plan acquisition interface which is based on the framework.

*This work is supported by the Air Force Systems Command, Rome Air Development Center, Griffiss Air Force Base, New York 13441-5700, the Air Force Office of Scientific Research, Bolling Air Force Base, District of Columbia 20332, under contract F30602-85-C-0008 and Ing. C. Olivetti & C., S.p.A..

1 Introduction

The majority of research in knowledge acquisition has concentrated on the acquisition of rules and concepts for rule-based expert systems [MMW85,KNM85,Boo84]. Domain-independent planners [CRM80] are another type of knowledge-based system whose performance in real environments depends critically on knowledge acquired from end users. Despite this, little attention has been paid to plan acquisition. The knowledge in a planning system is different both in its structure and its use than the IF-THEN rules in expert systems. The basic unit of plan knowledge, sometimes called a plan schema, contains information about the goal, preconditions, subgoals and effects of the action it represents. Figure 1 shows a plan schema from the POLYMER planning system developed at the University of Massachusetts [CL87]. It describes how to purchase an item and specifies causal dependencies, responsible agents, constraints, and affected objects as well as the goal and precondition information.

```
goal:            have(item,self)
decomposition:   fill-out(order-form), file(copy), mail(order-form),
                 receive(item), pay(invoice)
precondition:    have(money,self), have(order-form,self)
side-effects:    have-less(money.self), have(copy.self)
causal depend.:  enables(fill-out(order-form), file(copy)),
                 enables(fill-out(order-form), mail(order)),
                 enables(mail(order), receive(item)),
                 enables(receive(item), pay(invoice))
agents:          purchase-clerk
objects:         order-form, item, copy, money
constraints:     item-price < money-available
```

Figure 1: Purchase task as POLYMER plan.

To acquire plans, we have to carefully review the stages of knowledge acquisition, including explaining to a domain expert what we want to acquire,

eliciting the knowledge, coding the knowledge, displaying the knowledge, and debugging the knowledge. The psychology of programming [PK83] gives us a framework for defining the stages of plan knowledge acquisition and discussing the differences between acquisition for planners and for expert systems. This is done in the next section.

In order to build a plan acquisition system, we have to have a model that links the elicitable knowledge of the domain experts to the requirements of the planner. Having such a model allows us to design the plan acquisition system with some confidence that domain experts will be able to use it. We present a model for the recall of plans in section 3 and discuss the implications of the model in section 4. In section 5, we present the DACRON plan acquisition system which is based on the approach outlined. DACRON is designed to be used in conjunction with the POLYMER planning system, and issues such as the construction of plans, execution of plans, and handling of exceptions are POLYMER's responsibility. The main goal of DACRON is to allow domain experts to specify plan knowledge that can be used as a starting point, even if it is incomplete or incorrect. The final section discusses our future research plans.

2 Knowledge Acquisition: Planners versus Expert Systems

We view knowledge acquisition as an instantiation of general programming techniques. The psychology of programming can therefore be modified to accommodate knowledge acquisition [PK83]. Pea and Kurland identify four stages in the process of software development:

- understanding the problem;
- solving the problem;
- coding the solution;
- debugging the code.

The problem that has to be solved and coded in the general software environment by a programmer corresponds to a particular piece of domain

knowledge that has to be acquired, i.e. specified by a domain expert to the acquisition system. The programmer understanding the problem therefore corresponds to making the domain expert understand what particular knowledge the system wishes to acquire.

Solving the problem corresponds to the domain expert recalling a piece of knowledge. Gruber coined the term *acquirable form* for knowledge in this stage [Gru87].

The actual process of coding this knowledge (operationalization) means translating the acquirable form into a form meaningful to the system. In general programming this means transforming an algorithm to a programming language; in the case of knowledge acquisition it means expressing the acquirable form in terms of primitives of the acquisition system.

Once the knowledge is operationalized, it might be displayed in a different way than it was entered. This adds another stage to the process of knowledge acquisition. Users have to understand what they actually have specified. Only then can they understand how this new piece of information is viewed by the system, which in turn enables them to debug it.

We can summarize the stages of knowledge specification, which will serve to identify differences in the knowledge acquisition process between expert systems and planners:

- Request: Description of particular knowledge to be acquired;
- Elicitation: Recalling knowledge;
- Operationalization: Coding the recalled knowledge;
- Display: Echoing the coded knowledge;
- Debugging: Correcting the coded knowledge.

To acquire a rule, most existing knowledge acquisition systems for expert systems start at the first stage of this theory and follow the stages sequentially for each rule that is to be acquired. To acquire a single plan, domain experts have to move back and forth between these stages. Users might start the operationalization of the goal, move on to the request stage of the decomposition and then debug the precondition of a single plan.

Rules usually express a causal or temporal relationship. Plans usually consist of more elaborate information. Therefore units of knowledge are smaller in expert systems than in planning. This fact facilitates the request phase in expert systems.

Rule based expert systems have to elicit the `IF` part of the rule and the `THEN` part of the rule. Once the experts have recalled the knowledge that corresponds to each of these parts of the rule, they can start to specify it. Techniques used to accomplish this elicitation are usually based on association, comparison or cueing [Boo84,MMW85]. As plans have a more complex structure than rules (goal, subgoals, precondition, constraints, temporal and causal orderings, etc.), the elicitation process must be less uniform. Additional requirements for plan acquisition are a clear conceptual distinction between components of the plan, specific techniques to elicit particular components and an overall framework to integrate the components. Techniques developed for rule based expert systems, like repertory grids, heuristic classification, certainty ranking or choice annotation are of no help in this case. Certain forms of cued recall and structured frames seem to be most appropriate for the various demands in plan acquisition [MC88].

The largest problem in both areas is the operationalization of the elicited knowledge. The techniques applied most commonly are editors that specialize in the syntax of knowledge representation formalisms. This technique will not work as well in the acquisition of plan knowledge from domain experts because of the conceptually different slots of plans and the experts lack of knowledge representation skills. In planners, each slot needs to be addressed in a semantically correct way that is also immediately understandable to the domain expert. We found that electronic forms questionnaires as mentioned by Gruber and by Musen in conjunction with iconic representation of objects work best to fulfill all these requirements [Gru87,MFCS86].

Specified knowledge can either be displayed in the same form that it was entered or it can be set into the context of other knowledge in the system. For expert systems, this usually means displaying a rule network. The different reasoning mechanism in planners leads to a different display strategy, namely that of a hierarchy. Such a component is mandatory for successful debugging.

The above discussion shows that the acquisition of plan knowledge is considerably different than the acquisition of rule knowledge. In addition, there exists no model linking the humans subjective view of a task to a knowledge representation formalism. In this work, we try to build a plan acquisition system based on the implications of these differences.

3 A Framework for the Recall of Tasks

We would like to acquire plan knowledge from domain experts about the way they execute tasks. A task is an activity which is viewed by the human as a unit at a certain level of abstraction, e.g. *purchase an item* or *receive reimbursement for travel.* This focus eliminates many existing models of human cognition as possible candidates because either they make no predicitions on the task level [And83,AB73,Sch82], or they evaluate work done with computer systems themselves [CMN83,PK85] or they address interaction and discourse issues only [Suc84,Suc85,Sch75]. None of these theories makes statements about structures and processes involved in the recall of complex tasks from long term memory. Therefore, we started a series of interviews and experiments to build a framework for complex task recall. Interviewing domain experts about how they conduct certain complex tasks usually leads to a description of an example. In these examples, we observed the grouping of operations, which are the lowest level of description, into units. These units fit what is called *Handlung* by German psychologists and philosophers. A *Handlung* is a *conscious, goal-directed act of a human being, controlled by will, directed towards shaping reality. It contains three aspects: an intended goal, an analysis of means for its achievement and the decision to do so.*[1] A *Handlung* contains operations, conducted by a human, which transform states of reality into other states, serving a certain purpose [KB72]. (Clearly a *Handlung* is more complex than the English equivalent *act.* In the remainder of this paper we will refer to it as act though).

The act is the smallest coherent unit in the description of a task that appears to be at the appropriate level of abstraction. This individually perceived appropriateness as smallest unit, varying from person to person

[1]Definition from: Der grosse Brockhaus; 17th Edition. Translation by the authors.

and from task to task, makes the concept very suitable for our purpose. It also distinguishes the act from GPS-operators [NS72]. While operators are established by task analysis, acts are the representation of the human's perception of these tasks.

As the above definition of acts does not lend itself to immediate operationalization, we try to formalize its properties in information processing terms [SA77]. The representation of an act consists of four properties which are discussed below.

The first property of an act is the conscious goal; in our case it is the intention to complete a certain task. In our staged theory of knowledge acquisition, this intention is equivalent to understanding what is to be acquired. During work on one task, this goal remains the same, only situations (states) change, not intention. People consciously know about this goal and should be ready to report it without difficulty.

The current state of reality, as mentioned in the above definition [KB72], constitutes the second property. We call this property the pre-situation. During execution of the same task (same goal) the situation determines which operations are to be applied. Only when the task is completed or interrupted does the goal change. When people are actually performing a task, they directly perceive the pre-situation. During recall, the content of the working memory mirrors this state. People who are imagining working on a certain task should be readily able to report what the current state of affairs is.

The third property of the definition is the decision to generate behavior, which in turn is observable. We represent this as a list of names of the operations to be generated. As we are not concerned with actual execution of tasks, but only the part of them that can be reported by recall, we must be satisfied with the name the person ascribes to a certain operation. The mapping of theses names to primitives of the system is a question of coding and will thus be discussed in the next section.

The result is also available in the recalled act. We call this fourth property, describing the situation after the application of operations, the post-situation. The post situation is described in the same terms as the pre-situation but it includes the changes caused by the operations. The justification for this property comes from the explanation that acts transform states of reality into other states of reality.

Goal: Intention of the Whole Act
Pre-Situation: List of Properties
Operations: List of Operations
Post-Situation: List of Properties

Figure 2: Operationalization for acts

The complete representation of a formal act is given by the structure in figure 2. We are aware that the operationalization of such a complex concept as an act can not capture every nuance in meaning and must fall short in certain aspects of description. The structure we present here will certainly have to be amended. Acts are the product of recall of deeper cognitive structures. We assume that recall can be directed to decompose and sequentialize acts. Sequentialization is the process of finding a sequence of acts that leads from one state of the world to another one. Decomposition is the process of breaking an act into lower level acts, making the higher level act the goal and generating all its constituting operations as lower level acts. As our framework is concerned with the existence of recall processes and structures and not with low level, underlying cognitive processes, we make no assertions as to how this is accomplished. It is sufficient for our purposes to know that these processes and structures exist at all, not how and why.

In addition to the recall of activities, we assume the recallability of objects, relations and states of the world. As these entities comply to a certain degree with those items usually encountered in recall experiments, we need make no additional assumptions.

With the operationalization of acts and the proposed subprocesses, we were able to propose and test the following hypotheses about the recall of subject performed activities from long term memory. It is important to realize that these are only one set of many possible descriptions for the phenomena occurring but that they seem to be a reasonable one for our purpose.

1. A general task (goal) and a specific situation (pre-situation) will result in the recall of a specific set of operations.

2. Changing the pre-situation or the general task (goal) will result in the recall of different operations.

3. A goal, a start situation and an end situation will result in the recall of a sequence of acts. This sequence leads from the start situation over intermediate situations to the end situation.

4. An operation may itself be composed of acts and those acts are recallable (*decomposition*).

5. If the pre-situation of an act is not achievable, other acts with the same goal but different pre-situation can be recalled instead (alternative acts).

6. Operations can be distinguished from the post-situation created by these operations.

7. Causal dependencies between operations can be indicated.

To test these hypotheses, we first conducted a pilot study with four departmental secretaries and then a larger series of experiments. Our subjects in those experiments were 153 undergraduate and graduate students at the University of Massachusetts and Smith College. We could verify all hypotheses. Special care had to be taken with hypotheses number six (distinguishing post-situation from operations). Subjects were able to report the post-situation if asked directly about it. They could not report the post-situation if queried in general terms. For a detailed discussion of the experiments and their results see [MC88].

4 Implications of the Framework for the Acquisition of Plan Knowledge

The properties of the act framework can be used to create design criteria for the plan knowledge acquisition system.

The most important implication of the framework is that humans react to a situation differently with different intentions and that they do not set up recursive goal stacks but go step by step. People are guided by context and fine tuned by the current situation. In the explanation phase, a domain expert could be instructed to specify a certain piece of knowledge by just stating the goal of the plan.

The human description of a plan consists of a sequence of acts. This acquirable form can be directly translated from the elicitation phase to code. A plan definition system should allow the specification of each of these acts separately. The decomposition of acts into primitive plans and subgoals follows as a consequence of the primitives of the system. The act is either a primitive plan the system understands, or it has to be explained in terms of those primitive plans.

The problem of breaking a plan down into hierarchies of goals and subgoals should therefore not be forced on the user but evolve from the user's perspective. Decomposition and sequentialization are the processes to employ in this situation. The experiments show that a complex act can be decomposed. People can also give a sequence of acts from a start state to an end state. A combination of both techniques, decomposition and sequencing, seems to be most appropriate for the specification of complex plans.

People deal with one act at a time. The system should therefore not force the user to specify the sequence of acts as a whole. The specification of a new act should only begin when the last one is completely defined. This guarantees completeness and consistency of the individual pieces of knowledge and supports the human desire for closure.

The versatility of a planner can be greatly enhanced if there are numerous plans that achieve the same goal under different preconditions. People are able to report various acts to achieve a goal. The list of operations and the different preconditions are the major difference between these alternative acts. A plan specification system should solicit as much information as possible. It should therefore in the process of acquiring a plan also acquire alternative plans that meet the same goal.

Our experiments show that the specification of effects is more dependent on the interaction technique than others. People seem to assume that the effects of an operation or act are implicitly understood by the system.

Though people can answer all questions about effects correctly, they cannot report and specify them easily. An approach where users have to fill in forms or create forms yielded the best results.

The last property of the model was the indication of dependencies between operations. Our experiments show that though people think of and report their operations sequentially, they can indicate causal dependencies and thereby provide information for the parallel execution of plans. The specification of dependencies is best done graphically. Arrows indicating causal dependencies between acts are more easily understood than verbal or other methods.

5 The DACRON Plan Specification Interface for Domain Experts

DACRON is a plan acquisition system with an emphasis on immediate usability for the domain expert. DACRON addresses every aspect of the knowledge specification process introduced in section 2. We developed DACRON on the implications of the framework for task recall. Our goal was a system that would help domain experts to specify their knowledge in a way closest to their individual view of operational knowledge (acts) in the domain.

DACRON provides users with an *act type* to specify tasks. The act type appears as an icon. This icon can be opened and presents the user with a graphic form editor (figure 3). The editor shows compartments that refer to the pre-situation ("before-compartment"), the operations ("action-compartment") and the post-situation ("after-compartment") of the particular act, which correspond directly to the same entities in the framework. These compartments are ready to accept other icons as input.

The second construct is the *object/relation type*. Object/relation types represent objects and relations, which are used in the description of tasks. Object/relation types appear also as icons and can be opened to present their code in a form editor. The editor for object/relation types shows rows to hold names for arguments and primitive data types for arguments (figure 3). The icon for the object/relation type differs from that for the act type, as figure 3 shows, in that the act icon has extensions for arguments.

Icons carry a label to make them distinguishable.

To specify a new task, users can either copy and modify an existing act icon or begin with a blank one. Users start by naming the act type, e.g. *purchase-item-1* (figure 4). Then they can specify the goal by moving one or more object/relation types to the goal compartment and setting the arguments of these types to the desired goal-values, for example: moving the icon for *item* to the goal compartment, opening it and setting the value of *owner* to *self* (figure 4). Moving object/relation icons into the "before compartment" (pre-situation) and the "after compartment"(post-situation) and act icons into the "action compartment" (operations) specifies those slots of the plan. Causal dependencies between acts in the action compartment are indicated by arrows. Constraints are placed in the compartment that contains the object to which a certain constraint applies. In the case of inter-object constraints which might apply to objects in different compartments the users might choose the more convenient one.

Another implication of the framework was the decomposition of the goal of a general task into a sequence of acts. As the granularity of these acts is dependent on the view of the domain expert, DACRON does not enforce a decomposition hierarchy but lets the users follow their own decomposition path. The users give the sequence of acts at a level they think is appropriate. The users do this by specifying an act type. DACRON guarantees completeness of this specification by prompting the user for unspecified compartments.

Every item in the knowledge base is represented by an icon. A permanent window, called the *archive*, presents at any given moment a part of the knowledge base (figure 5). To see different parts of the knowledge base, the archive can be moved over the knowledge base, which is represented as a two-dimensional plane filled with icons. In addition, users may zoom in and out on the knowledge base.

To reorganize the clustering of icons in the knowledge base, users may employ the *retriever*. The retriever allows users to order the icons in the knowledge base by certain keys like name, date of creation, etc. or to search for icons by goal, constituent parts, etc. These retriever commands can be invoked by opening a special icon that is always present Upon opening this icon, users can pick commands from a menu and provide the desired options. The *help* icon is close to the *retriever* and provides general and

specific help functions such as animated introduction to the system and help with special commands and tasks.

DACRON has two other facilities which are intended to aid users at the display stage and the debugging stage of the knowledge acquisition process. A *debugger* can be invoked to give an animated demonstration of the specification process of a type. The animation shows how the type was created, in which order arguments were given, what icons were placed into the type, etc. As this debugger is interactive, it is possible to stop it at any given moment and change the specification of the type shown. The animation can also be used for training purposes, teaching new users how DACRON is used.

The second facility is the *reviewer*. It is used to present sequences of plans and alternatives at crucial points in the planning process to the users. In this situation, we use the DACRON interface not only for the specification of activities, but also for the presentation of the planning process in POLYMER.

6 Future Work

We currently have the display components and rudementary acquisition components of the DACRON system implemented. We can show existing objects and activities in the DACRON-format. In the nearer future we will concentrate on the completeness of the acquisition component. Later work will be devoted to the animated presentation of planning processes and the use of animation in debugging the description of activities. We are also investigating the use of color as a coding technique for constraint definition.

We are planning to test and evaluate DACRON with users in a local office environment. These user studies would involve the specification of office tasks by clerks and secretaries using DACRON and at a later point the display of whole office activities to office workers in the process of accomplishing complex tasks.

References

[AB73] John R. Anderson and Gordon H. Bower. *Human Associative Memory.* Winston and Sons, 1973.

[And83] John Robert Anderson. *Architecture of Cognition.* Harvard University Press, 1983.

[Boo84] J. Boose. Personal construct theory and the transfer of human expertise. In *Proceedings of the national conference on artificial intelligence*, Austin, Texas, 1984.

[CL87] Bruce Croft and Larry Lefkowitz. A goal-based representation of office work. In *Proceedings of the IFIP workshop on office knowledge*, Toronto, 1987. North Holland.

[CMN83] Stuart K. Card, Thomas P. Moran, and Allen Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum, 1983.

[CRM80] Eugene Charniak, Christopher Riesbeck, and Drew McDermott. *Artificial Intelligence Programming.* Lawrence Erlbaum, Hillsdale, 1980.

[Gru87] Thomas Gruber. Acquiring strategic knolwedge from experts. In *Proceedings of the knowledge acquisition for knowledge based systems workshop*, pages 10.0–10.18. American Association for Artificial Intelligence, 1987.

[KB72] G. Klaus and M. Buhr. *Philosophishes Woerterbuch.* VEB Deutscher Verlag der Wissenschaften, Ost-Berlin, GDR, 1972.

[KNM85] G. Kahn, S. Nowlan, and J. McDermott. MORE: An intelligent knowledge acquisition tool. In *Proceedings of the ninth international conference on artificial intelligence*, Los Angeles, California, 1985.

[MC88] Dirk E. Mahling and W. Bruce Croft. Relating human knowledge of tasks to the requirements of plan libraries. Technical Report 88-33, University of Massachusetts at Amherst, 1988.

[MFCS86] Mark Musen, Lawrence Fagan, David Coombs, and Edward 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.

[NS72] Allen Newell and Herbert Simon. *Human Problem Solving.* Prentice-Hall, 1972.

[PK83] R. D. Pea and D. M. Kurland. On the cognitive prerequisites of computer programming. Technical Report TR 18, Bank Street College New York, 1983.

[PK85] P. G. Polson and D. E. Kieras. A quantitative model of the learning and performance of text editing knowledge. In L. Borman and B. Curtis, editors, *Human Factors in Computing; CHI '85.* ACM, Inc., New York, 1985.

[SA77] R. C. Schank and R. P. Abelson. *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum, 1977.

[Sch75] C. F. Schmidt. Understanding human actions. In *Conference on Theoretical Issues in Natural Language Processing*, 1975.

[Sch82] Roger C. Schank. *Dynamic Memory.* Cambridge University Press, 1982.

[Suc84] Lucy A. Suchman. Office procedures as practical action. *ACM Transaction on Office Information Systems*, 4:320–328, 1984.

[Suc85] Lucy A. Suchman. Plans and situated actions: The problem of human-machine communiation. Technical Report isl-6, Xerox Corporation, 1985.

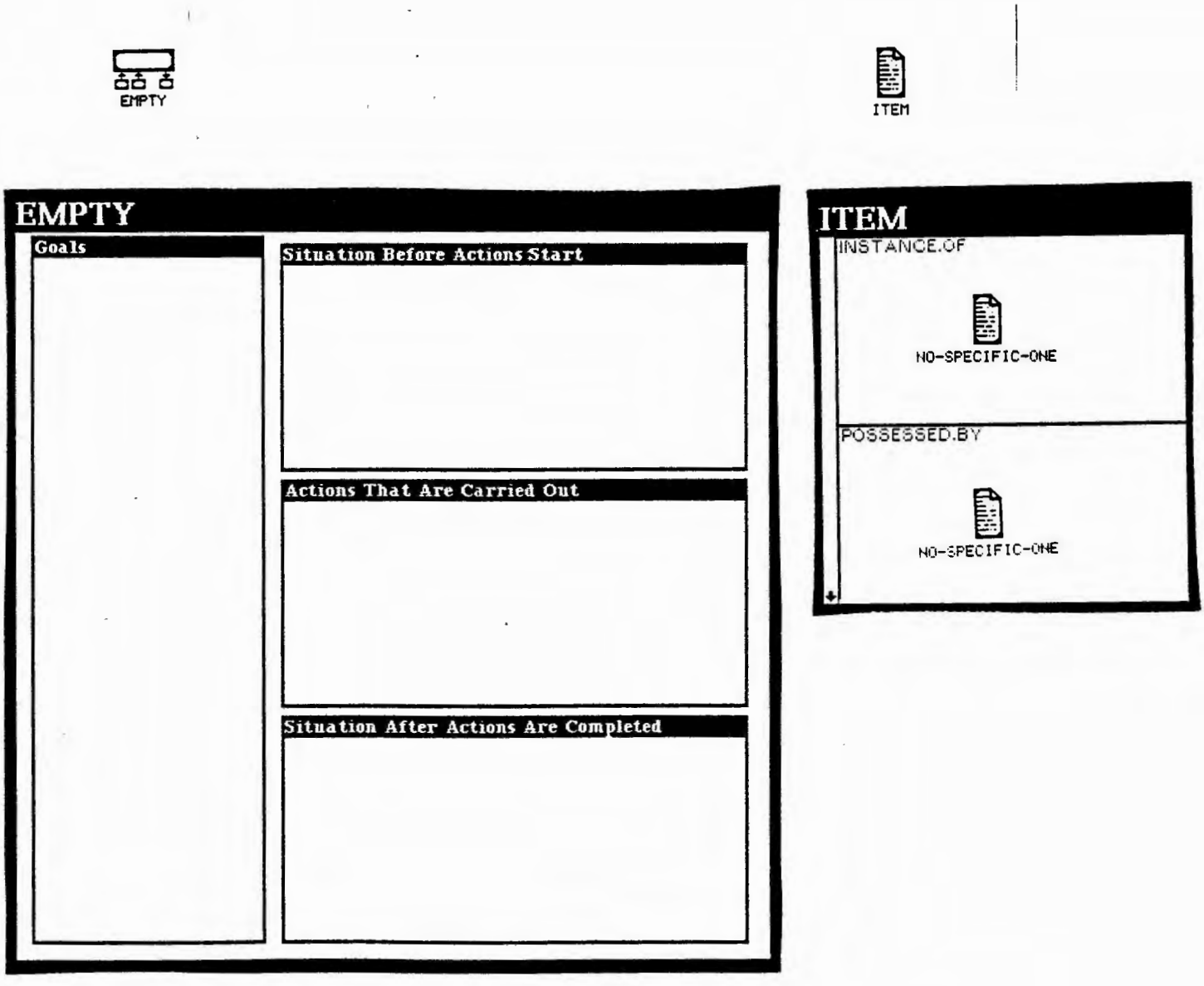


Figure 3: Icon and box representation for act type and object/relation type

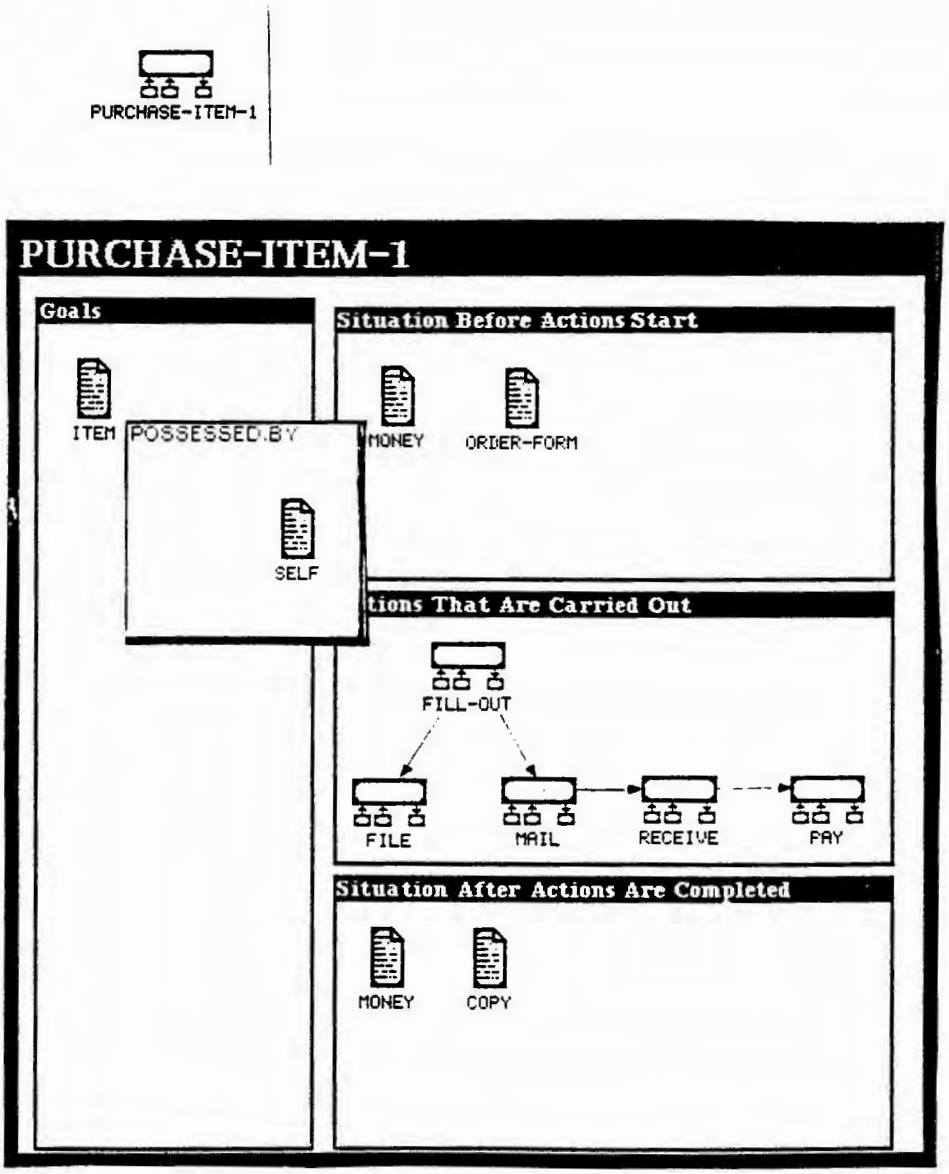


Figure 4: Icon and box for the purchase-task (with goal-value set).

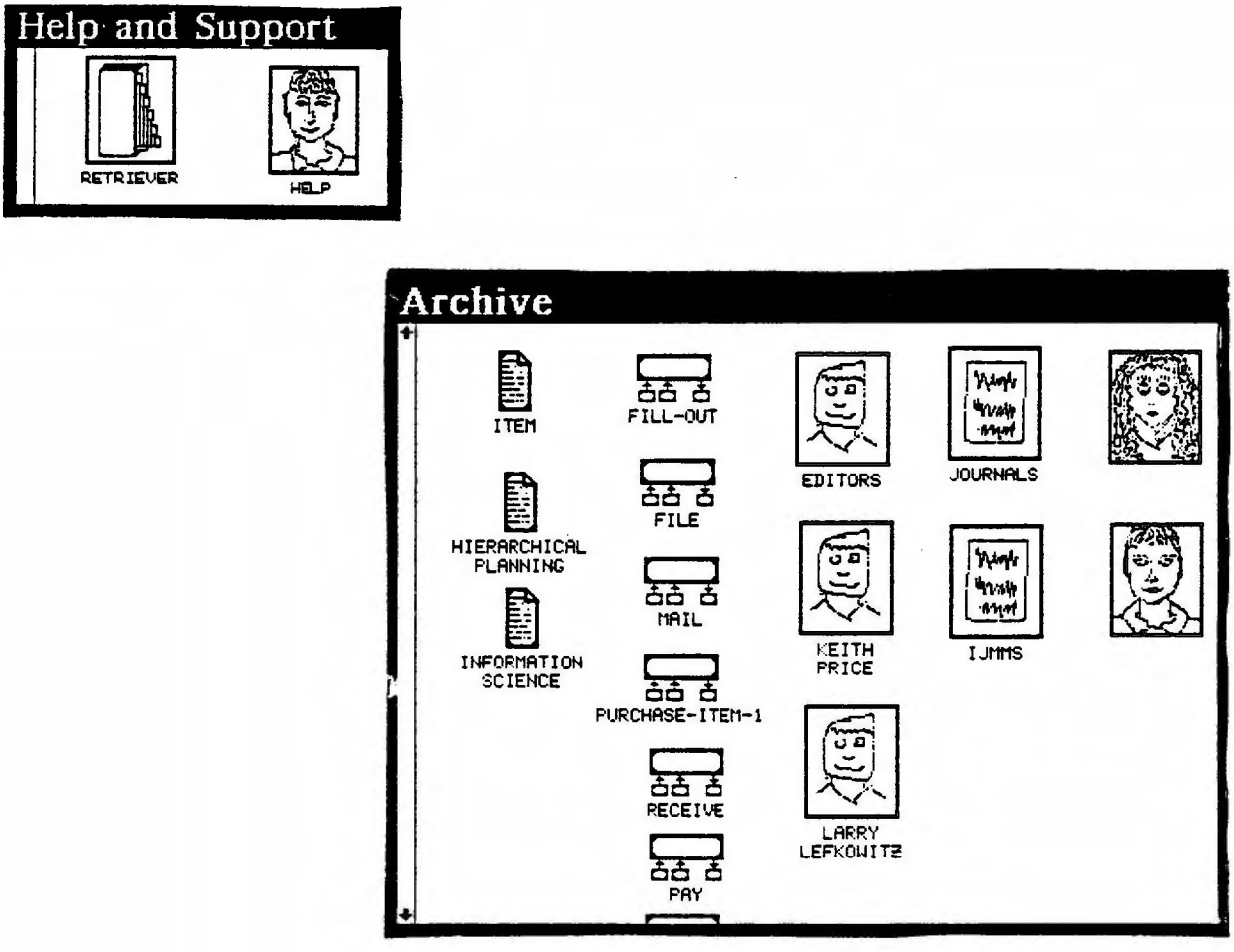


Figure 5: Archive

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Understanding Subtasks from a Piecemeal Collection of Knowledge

Sandra Marcus
Advanced Technology Center
Boeing Computer Services
P.O. Box 24346, MS 7L-64
Seattle, WA 98124

Design experts are good at identifying necessary parts of a design, such as the components of an artifact, and detailing the considerations that go into determining the parameters of individual parts. While they are good at performing design, it is more difficult for them to describe a complete strategy for building a design from the individual constraints. SALT is an automated knowledge acquisition tool for constraint-satisfaction tasks that employs a very promising set of representational primitives. SALT's level of representation along with its piecemeal, bottom-up approach to building the knowledge base was chosen because it capitalizes on what the expert can describe best and provides help where the expert needs it most. This paper describes a proposal for helping experts articulate high level strategies for deciding what pieces of the design should be looked at together and the order in which to attack subparts of the design.

1. Introduction

Acquiring knowledge from human experts is a major problem when building a knowledge-based system. It is a very expensive, labor-intensive process that limits the number of systems that can be produced with fixed resources. In addition, domain experts typically have difficulty in expressing their problem-solving knowledge explicitly. Explanations may be in terms of idealized verbal descriptions learned in school, while much expertise seems to be compiled in patterns that are accumulated over the years. It can be very difficult for the expert to talk about these compiled patterns.

This is especially true in the area of design. Human designers and schedulers are fairly good at describing individual considerations for constructing a solution. They can extemporaneously list many of the constraints that the solution needs to satisfy. They can consult manuals of formulas and tables for producing values for individual

design parameters. But they are less clear on how the individual considerations should fit together in a step-by-step procedure for producing a design. The main difficulty is that decisions affecting one part of a design often impact other parts of the design in complex ways. It is very hard for human problem solvers to describe what the complex interactions are and how to keep them in mind while making decisions. The knowledge acquisition problem is compounded when knowledge is added as a system evolves over time or when a system must represent expertise from multiple experts, each expert in a different part of the problem.

SALT (Knowledge Acquisition Language) is an automated knowledge acquisition tool for constraint-satisfaction tasks such as designing an artifact or constructing a production schedule [Marcus 85, 86]. SALT acquires knowledge from an expert and generates a domain-specific knowledge base compiled into rules. This compiled knowledge base is then combined with a problem-solving shell to create an expert system. SALT maintains a permanent, declarative store of the knowledge base which is updated during interviews with the domain expert, analyzed for completeness, consistency and soundness and used as the input to the compiler. This approach has been field-tested by Westinghouse Elevator Company [Marcus 87]. Westinghouse has used SALT to build and maintain VT (Vertical Transportation), a 3000-rule system for designing elevator systems that is in production use.

SALT's knowledge base representation language identifies domain knowledge by its function in problem solving and has the following desired properties [Marcus 88]:

- The language provides the basis for well-motivated knowledge elicitation strategies. It allows SALT to help the user identify relevant domain knowledge.

- Analyses run on the intermediate store of the knowledge base can help determine the adequacy of the knowledge base.

- The language identifies knowledge by its function in problem-solving. It therefore is a good key to how and when the knowledge should be used by the problem solver.

- The language provides the basis for an explanation facility that describes what is in the knowledge base and can justify the generated system's reasoning.

- The knowledge base representation identifies important features of the program that should be tested in any validation study. The generated expert system can analyze a set of test cases for their coverage of these features.

SALT has demonstrated advantages using its representational primitives and a bottom-up approach to building a knowledge base. However, when a SALT-generated expert system uses the pieces of domain knowledge collected, additional strategies for what parts of the design to work on in what order may help the system reach a solution faster. Domain-independent strategies for efficient search alone can't help. It is important to take into account the fact that some solutions will be preferred over others based on domain-specific considerations such as cost, performance or manufacturability. SALT's knowledge base representation can be helpful in forming useful strategies.

Section 2 below describes SALT's piecemeal knowledge elicitation strategy and the representational scheme used to put the pieces together. It describes some current facilities for checking the soundness of the knowledge base and the way individual considerations are compiled in the generated expert system. Section 3 identifies some design issues for compiling a domain-specific knowledge base as SALT represents it. Section 4 describes a proposal to use the SALT representational scheme to facilitate the creation of higher-level strategies for applying the knowledge pieces SALT currently collects.

2. SALT's bottom-up creation of a knowledge-based system

Like many other successful knowledge acquisition tools [Davis 77, Boose 84, Kahn 85, van de Brug 86, Eshelman 86, Klinker 87], SALT gets its power by making a strong assumption about the problem-solving strategy that will be used by any expert system it creates. SALT assumes a generic propose-and-revise problem-solving

strategy for creating a design. SALT-generated systems start by proposing values for design parameters, one parameter at a time. As in planners like MOLGEN [Stefik 81a, 81b], the system tries to delay making a decision until all factors affecting that decision are known. But it may not always be possible to do this. In such cases the system uses domain knowledge to guess; it proposes a plausible, preferred value given what it knows at that point. The generated system also identifies constraints on design parameters, and proposed values are checked against their constraints. If a constraint violation in the proposed design is detected, the system considers suggestions for revising proposed design parameters. Once a revision is decided on, a truth maintenance system [Doyle 79] removes any part of the proposed design dependent on the old value, and the process of extending the proposed design resumes.

Because SALT assumes a specific problem-solving strategy, it can represent domain-specific knowledge according to the role it will play in finding a solution. The knowledge role identifies what constitutes a useful knowledge piece, specifies how a knowledge piece interacts with other pieces, and determines how it should be used in the generated expert system.

2.1 Acquiring a piece of design knowledge

Given that the assumed problem-solving strategy is to incrementally extend a proposed design, check constraints on design parameters and revise decisions in response to constraint violations, there are three major roles that domain-specific knowledge can play in the system:

(1) PROPOSE-A-DESIGN-EXTENSION
(2) IDENTIFY-A-CONSTRAINT on a design parameter
(3) PROPOSE-A-FIX for a constraint violation

SALT aids users by allowing them to enter knowledge piecemeal starting at any point. The grain size of the pieces corresponds roughly to the SALT representational primitives: Users can supply a procedure for specifying a parameter value, can identify a constraint on a parameter value or can suggest a remedy for a constraint violation. SALT keeps track of how the pieces are fitting together and warns the user of places where pieces might be missing or creating inconsistencies.

The examples shown below are based loosely on the VT knowledge base. VT takes as input customer requirements, such as how fast the elevator should travel and what its carrying capacity should be, as well as architectural details about the building it will service, such as floor heights and wall-to-wall dimensions in the elevator shaft. VT must produce a list of quantities, ordering codes and other parameters for all equipment required, including some routine modifications of standard equipment, and an equipment layout customized to the elevator shaft.

In any SALT interview, the user starts by identifying the kind of knowledge he or she wishes to enter by selecting from the menu below:

```
1.  PROCEDURE          Enter a procedure for a value
2.  CONSTRAINT         Enter constraints on a value
3.  FIX                Enter remedies for a constraint violation
4.  EXIT               Exit interviewer

Enter your command [ EXIT ]:
```

PROCEDURE is used to enter knowledge to PROPOSE-A-DESIGN-EXTENSION, CONSTRAINT for IDENTIFY-A-CONSTRAINT knowledge and FIX for PROPOSE-A-FIX knowledge. For each type of knowledge, SALT displays a schema that prompts for the detailed information needed to apply knowledge of that type. The user fills in the slots of the schema. An example of a completed schema for a PROCEDURE for determining a value for MACHINE-GROOVE-PRESSURE is shown below with prompts on the left and knowledge supplied on the right:

Knowledge Piece (1)

```
1  Name:                  MACHINE-GROOVE-PRESSURE
2  Precondition:          NONE
3  Procedure:             CALCULATION
4  Formula:               SUPPORTED-LOADS /
                           [ [ MACHINE-SHEAVE-DIAMETER *
                           .5 ] * HOIST-CABLE-QUANTITY ]
5  Justification:         THIS PROCEDURE IS TAKEN FROM
                           PARTS MANUAL II, P. 388.
```

The user is instructed to supply a procedure that includes all of the necessary considerations that determine the value for the variable. Here the user indicates

that MACHINE-GROOVE-PRESSURE should be calculated with the formula supplied using the values of SUPPORTED-LOADS, MACHINE-SHEAVE-DIAMETER, and HOIST-CABLE-QUANTITY. The absence of a precondition means that this is the procedure that should be used for all jobs. The justification documents the reason for using this formula.

The schema for CONSTRAINT is similar in that the user must supply a procedure for determining the value of the constraint. In addition, the schema prompts for identification of the value constrained and a specification of the kind of constraint it is. For example, the value of MAXIMUM-MACHINE-GROOVE-PRESSURE might be identified as placing a MAXIMUM on the value of MACHINE-GROOVE-PRESSURE.

An example of a schema for a FIX knowledge piece is shown below:

Knowledge Piece (2)

1	**Violated constraint:**	**MAXIMUM-MACHINE-GROOVE-PRESSURE**
2	**Value to change:**	**HOIST-CABLE-QUANTITY**
3	**Change type:**	**INCREASE**
4	**Step type:**	**BY-STEP**
5	**Step size:**	**1**
6	**Rating of undesirability:**	**4**
7	**Reason for undesirabiity:**	**Changes minor equipment sizing**

Each piece of FIX knowledge is focused on a remedy for a single constraint violation, here MAXIMUM-MACHINE-GROOVE-PRESSURE. If this constraint is violated by the proposed design, the expert suggests an INCREASE in the value of HOIST-CABLE-QUANTITY BY-STEPs of 1. The expert may suggest more than one fix for a given constraint violation. For example, the expert might also suggest a DECREASE in COMP-CABLE-UNIT-WEIGHT, which would reduce SUPPORTED-LOADS. Rating of undesirability is used to determine preference among fixes.

The rating of undesirability is selected from a domain-specific ordering of negative effects that design modifications can have. The one developed by the VT domain expert is shown below:

```
1.  Causes no problem
2.  Increases maintenance requirements
3.  Makes installation difficult
4.  Changes minor equipment sizing
5.  Violates minor equipment constraint
6.  Changes minor contract specifications
7.  Requires special part design
8.  Changes major equipment sizing
9.  Changes the building dimensions
10. Changes major contract specifications
11. Increases maintenance costs
12. Compromises system performance
```

Relative position on this scale is relevant but absolute position is not. This is because not all of the negative effects can be measured in equivalent units, like dollars and cents.

The domain expert starts by entering individual knowledge pieces like these. SALT links these pieces together in its declarative representation of the knowledge base and uses its analyses of how the pieces fit together to prompt for additional information.

2.2 Linking the pieces in the knowledge base

SALT's representational scheme is built around the framework of a dependency network. Each node in the network is the name of a variable that is assigned a value when the expert system runs; this can be the name of an input, a design parameter, an intermediate value used to determine a design parameter or a constraint. There are three kinds of directed links that represent relations between nodes: (1) "Contributes-to" will link *A* to *B* if the value of *A* is used in a procedure to specify a value for *B*. (2) "Constrains" will link *A* to *B* if *A* is a constraint and the value of *A* places some restriction on the value of *B*. (3) "Suggests-revision-of" will link *A* to *B* if *A* is a constraint and a violation of *A* suggests a change to the value of *B*. Each of these links is supported by additional information in the knowledge base: (1) "Contributes-to" links are supported by details of how contributors are combined to specify the value of the node pointed to. (2) "Constrains" links are supported by a specification of the nature of the restriction. (3) "Suggests-revision-of" links are supported by a declaration of the nature of the proposed revision (e.g., direction and amount of change) and its relative preference.

An example of an instantiated scheme based loosely on the VT knowledge base is shown in Figure 1. Each node in Figure 1 gives the name of a value that the expert system should be able to generate. The "contributes-to" links from HOIST-CABLE-QUANTITY, MACHINE-SHEAVE-DIAMETER and SUPPORTED-LOADS to MACHINE-GROOVE-PRESSURE are supplied by Knowledge Piece (1) above. The "suggests-revision-of" link from MAXIMUM-MACHINE-GROOVE-PRESSURE to HOIST-CABLE-QUANTITY is given by Knowledge Piece (2).

This level of representation along with the piecemeal, bottom-up approach to building the knowledge base was chosen because it capitalizes on what the expert knows best and provides help where the expert needs it most, where human memory limitations make it hard to keep track of all the interconnections among decisions. SALT analyzes the connections and helps the user keep track of them.

For example, a user may start by supplying a procedure for determining a value for a particular design parameter. If this procedure uses other values that the system needs to generate, SALT will keep track of these contributors. At the end of the session (or at the beginning of the next session, if desired), SALT will remind the user of any contributors for which the user has not supplied a specification procedure.

SALT detects any loops in the dependency network (that is, loops along "contributes-to" links). For example, suppose the value of SUPPORTED-LOADS determined the value of HOIST-CABLE-QUANTITY. As shown in Figure 1, the value of HOIST-CABLE-QUANTITY is needed to generate a value for SUPPORTED-LOADS. In such a case, SALT guides the domain expert in how to use propose-and-revise to get values for nodes in this part of the network. The user is asked to supply a procedure for getting a plausible, preferred estimate for some value on the loop. The original procedure is used by the expert system to calculate a constraint on the estimate so the user is asked what kind of limit it places on the estimate. Then the user is asked to provide ways to revise the estimate if this constraint on it is violated.

In general, SALT reminds the user to provide suggestions for revisions in response to violations of any constraint mentioned in the course of an interview and uses the dependency network to suggest candidate remedies. Any contributor to a constraint or the value it constrains, even indirect ones, can have an effect on their

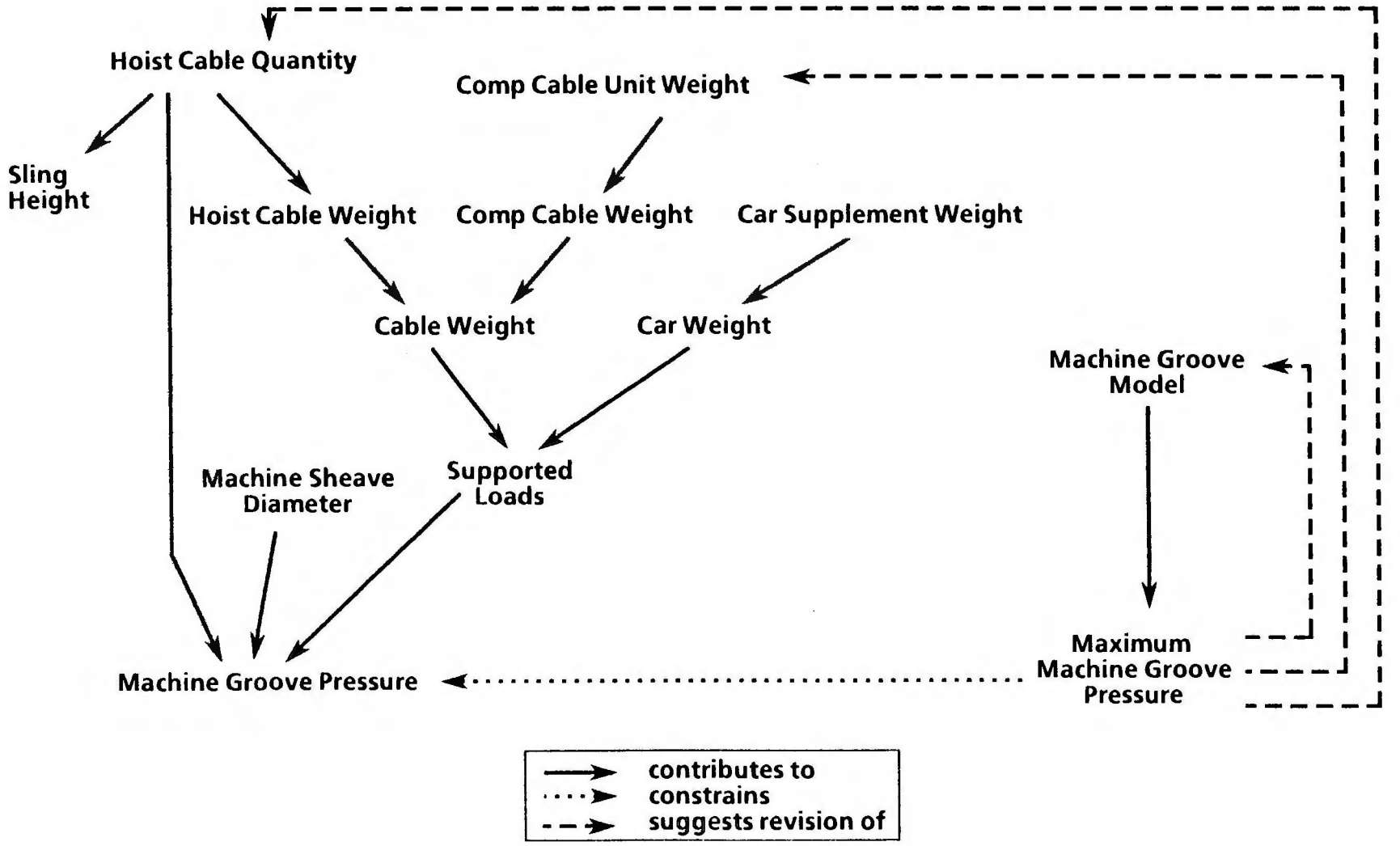


Figure 1: Example of the knowledge base relations represented by SALT.

relationship. Thus, depending on the nature of the contribution, any contributor might serve as a helpful revision point and SALT reminds the user what these are. It is up to the user to supply domain knowledge about what is feasible and desirable to modify.

SALT also analyzes the dependency network to detect potential interactions among revisions that might cause the system to thrash. Thrashing will occur if a fix for one constraint violation makes it harder to satisfy a second constraint whose fixes in turn make it harder to satisfy the first constraint. A generated expert system that only paid attention to local constraint violations might get into an infinite loop going back and forth between the two constraint violations. SALT uses identification of areas of potential constraint antagonism in the network to compile a more global view of the fixes involved. The next section describes the compilation procedure for the current version of SALT.

2.3 Compiling the pieces into an expert system

For SALT, compiling the declarative, domain-specific knowledge base requires specifying when each piece of knowledge should be brought to bear. For incrementally extending the proposed design and identifying constraints, the generated system uses data-driven control; a variable (e.g., representing a design parameter or a constraint) can be assigned a value whenever values for its contributors are available. If more than one variable's procedure is ready for use at the same time, the SALT compiler allows a random choice on which should be used first. The way knowledge gets used in this portion of the problem solving is fairly close to the way it is entered by the expert.

For managing revisions in response to constraint violations, the SALT compiler uses analyses of the boundaries of fix effects to control knowledge use. When the generated system encounters a constraint violation whose fixes don't cause trouble elsewhere, it will work on that violation until it is fixed. It will explore fixes, and combinations of fixes, for that constraint in order of decreasing preference given by the expert's rating of undesirability. The system will contain the design extension to only those nodes that will allow it to tell whether the constraint violation is remedied before building on those values. For example, for the knowledge base in Figure 1, the relevant area has been shaded in Figure 2. Once the system starts

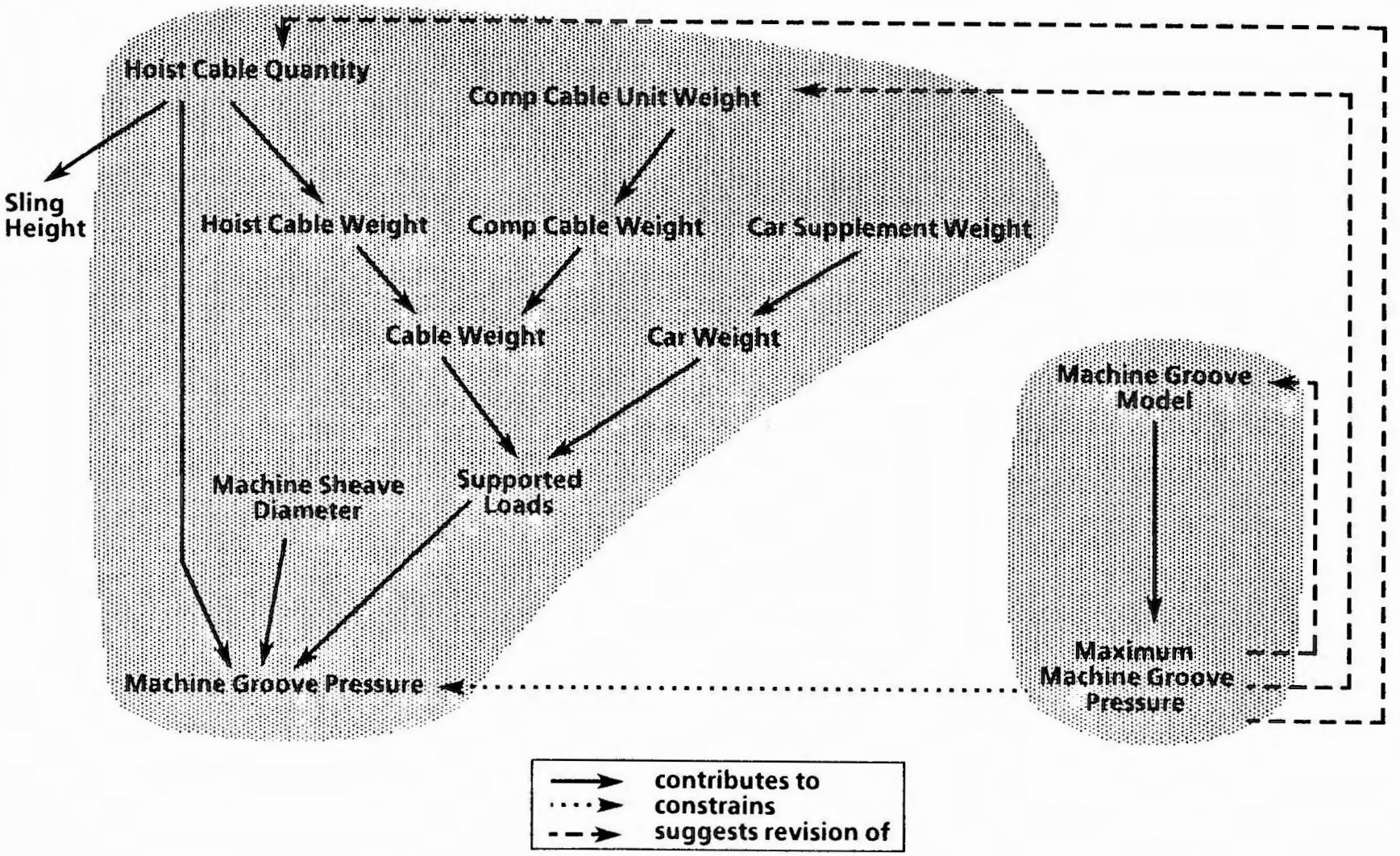


Figure 2: Knowledge base shaded to show the part needed to evaluate satisfaction of the Maximum Machine Groove Pressure constraint.

working on fixing MAXIMUM-MACHINE-GROOVE-PRESSURE, it will stay within the shaded area and prevent, for example, the assignment of a value of SLING-HEIGHT and its dependents until MAXIMUM-MACHINE-GROOVE-PRESSURE is fixed.

For areas of the network that are identified as having potentially antagonistic constraints, the search is broadened to include the area of influence for fixes of all the antagonistic constraints. The system will try all possible combinations of fixes for the entire set of constraints. Figure 3 expands on Figure 1's knowledge base by adding a second constraint-constrained pair. In order to make the figure more readable, "suggests-revision-of" links have been omitted. Instead, suggested revisions in response to a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE are surrounded by rectangles while suggested revisions for violations of MAXIMUM-TRACTION-RATIO are enclosed in ovals.

SALT identifies this area of the knowledge base as containing potentially conflicting constraints. For example, a violation of MAXIMUM-TRACTION-RATIO calls for an increase of COMP-CABLE-UNIT-WEIGHT, a decrease of CWT-TO-PLATFORM-DISTANCE and/or a downgrade of MACHINE-GROOVE-MODEL. A violation of MAXIMUM-MACHINE-GROOVE-PRESSURE calls for a decrease of COMP-CABLE-UNIT-WEIGHT, an upgrade of MACHINE-GROOVE-MODEL and/or an increase of HOIST-CABLE-QUANTITY. Fixing a violation of MAXIMUM-TRACTION-RATIO by changing COMP-CABLE-UNIT-WEIGHT or MACHINE-GROOVE-MODEL will make it harder to fix a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE and vice versa.

Suppose that the expert system began by specifying the values contained in the cluster labeled "1." Before generating other values that MACHINE-GROOVE-PRESSURE might contribute to, or before following the offshoot of HOIST-CABLE-QUANTITY to SLING-HEIGHT and so on, the problem solver should evaluate whether MACHINE-GROOVE-PRESSURE meets its maximum. Thus, the cluster labeled "2" should be generated next. Finally, since revisions for MAXIMUM-TRACTION-RATIO can interact with revisions for MAXIMUM-MACHINE-GROOVE-PRESSURE, the clusters labeled "3" should be handled after that. The clustering is simply based on identifying contributors to constraint evaluations. The ordering is conditional; once one constraint of an interacting pair has been identified, the other should be evaluated to prevent building on values that might be changed. Keeping track of

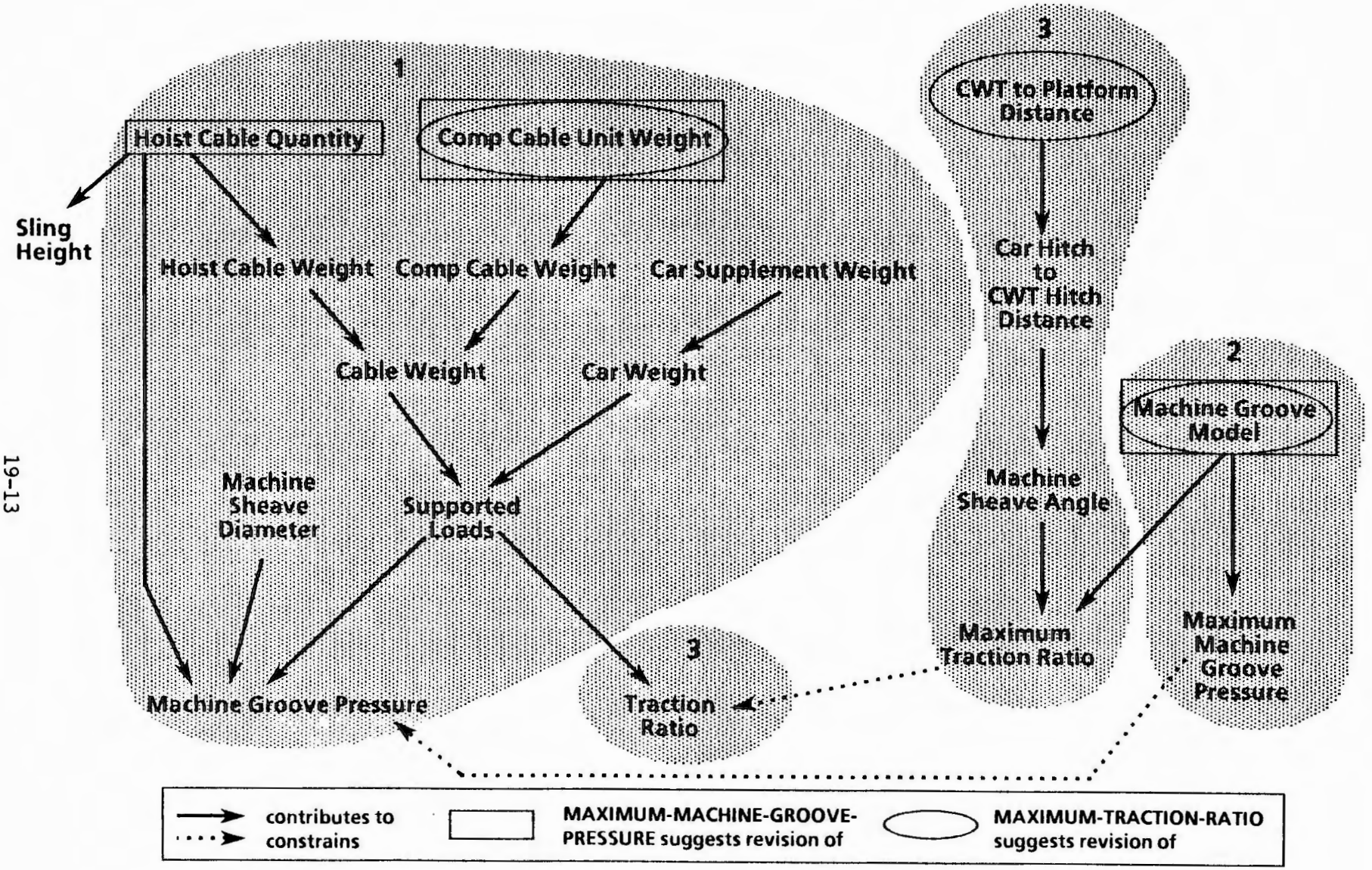


Figure 3: Subtask ordering to work on Maximum Machine Groove Pressure and then address Maximum Traction Ratio.

combinations of fixes for both potentially conflicting constraints will prevent thrashing.

3. Issues in the Design of the Compiler

In developing a compiler for an expert system, there are two important goals to achieve: (1) The solution reached should be optimized; that is, it should be a preferred solution in domain-specific terms such as safety or dollar cost or customer satisfaction. (2) A solution should be reached within a reasonable amount of time. Linear programming has provided algorithms for reaching optimal solutions in cases where preference can be quantified. Such quantification is not available in domains SALT is intended for, like VT's. When one choice is preferred over another for different reasons, such as cost versus performance, it is hard to specify the units of degree of preference. In another class of constraint-satisfaction problems, substantial work has been done to develop domain-independent algorithms for efficiently ordering value assignments [Freuder 82, Dechter 85, 87]. In these problems, generation of any solution which satisfies all constraints constitutes success. This is not true in domains for which SALT is intended, like VT's.

For SALT-generated expert systems, domain-specific preference considerations directly affect the decision ordering; SALT-generated systems will try for plausible, preferred solutions first. The domain expert expresses preference and some implicit ordering heuristics in a number of ways. For example, suppose the expert knows that the following relationship between parameters is required: $A + B$ must be less than or equal to C, where C is derived independently from A and B, and both A and B have independent minimums placed on them. Should the system (1) estimate A and calculate B using $C - A$, (2) start with B and calculate $A = C - B$ or (3) estimate both A and B, take their sum and compare the sum to C? In all three cases, constraints will be checked and which constraint is violated will influence whether the system will modify the values of A, B, C, their contributors, and/or their constraints.

The expert must decide how to cast the problem, assigning pieces of knowledge to the roles of proposing a value for a design parameter, identifying a constraint or suggesting a remedy for a constraint violation. Since during problem-solving,

proposing a design parameter will occur before checking constraints on it, which will occur before revision in response to violation of constraints on it, some decision ordering is acquired from the expert's choice in assigning knowledge roles. As mentioned above, SALT tries to help the user make these knowledge role assignments by letting the user see the potential effects of various decisions. Users are cautioned that what goes into each category should be based on what tends to produce an optimal elevator design; i. e. , the most preferred values should be given to propose a design extension and less desirable ones as explicitly ordered revisions.

Beyond the ordering implicit in assigning knowledge to the three knowledge roles and explicit in rating potential revisions, no other ordering information is currently acquired from the expert. In particular, when extending a design by proposing values for design parameters, ordering is completely data-driven. In some cases, the control derived from domain-specific constraints and preferences is under-determined; multiple decision orderings may be possible. Other shells and problem-solving systems that perform tasks similar to the ones SALT is intended to acquire employ efficiency strategies in ordering decisions to propose design extensions. These strategies are hand-coded by the system designers of systems such as MPA [Herman 86] written in DSPL [Brown 85], OPIS [Smith 86] and PRIDE [Mittal 86]. There is a potential for SALT either to generate similar goal structures in compiling the knowledge base or to support a user in describing top-level strategies by supplying useful proposals and analyses.

4. Creating strategy from the SALT representation scheme

It is clear that organizations that impose orderings on the way tasks are addressed can improve the performance of a problem solver (cf., for example, [Smith 86]). In a propose-and-revise system, a good use of ordering should minimize the number of times backtracking is initiated and minimize the amount of work that has to be undone and redone when backtracking is required. However, expecting an expert to produce such orderings on a clean slate is probably asking too much. If the expert gives the kind of bottom-up description of the domain required by SALT, many of the considerations that should affect ordering will be extracted from the user in the normal course of the interview and will be available in SALT's intermediate knowledge base store. The potential is then there for SALT either to

generate its own goal structure in compiling the knowledge base or to support a user in describing top-level strategies by supplying useful proposals and analyses.

SALT already does the analysis to determine what design decisions affect others. These are the analyses that produce the clusters shown by shading in Figures 2 and 3 that are used in containing revision activity. The same analyses can be used to group decisions in proposing design extensions and identifying constraints. Decisions within one cluster should be completed before moving on to another cluster.

SALT also predicts likely places of constraint interaction. In cases where decisions interact, by recognizing priority of some clusters over others, the search for a solution can be speeded up. Figure 4 shows an alternate subproblem clustering and ordering for the same network shown in Figure 3. In Figure 4, the evaluation of the MAXIMUM-TRACTION-RATIO is addressed before that of the MAXIMUM-MACHINE-GROOVE-PRESSURE. The following sections describe two criteria that can be useful in choosing between the alternate plans represented by Figures 3 and 4: (1) Start with the constraint most likely to be violated. (2) Start with the constraint with the most limited resources for satisfaction. The sections below give an example in which these strategies might be useful and an indication of how SALT could help a user craft such a strategy.

4.1 Start with the constraint most likely to be violated

Starting to work on the constraint most likely to be violated will cut down on the re-work that has to be done when proposed values are modified and work that depended on old values is erased. In the VT domain, if CAR-WEIGHT is unusually high, it is more likely that MAXIMUM-MACHINE-GROOVE-PRESSURE will be violated by the proposal so the plan in Figure 3 should be used. If CAR-WEIGHT is unusually low, it is more likely that MAXIMUM-TRACTION-RATIO will be violated and the plan in Figure 4 should be used.

SALT can help the expert develop a strategy like this by asking the expert for a kind of knowledge-based sensitivity analysis. SALT can identify the relevant task clusters and display the contributors to the constraint subproblem. SALT can then ask the

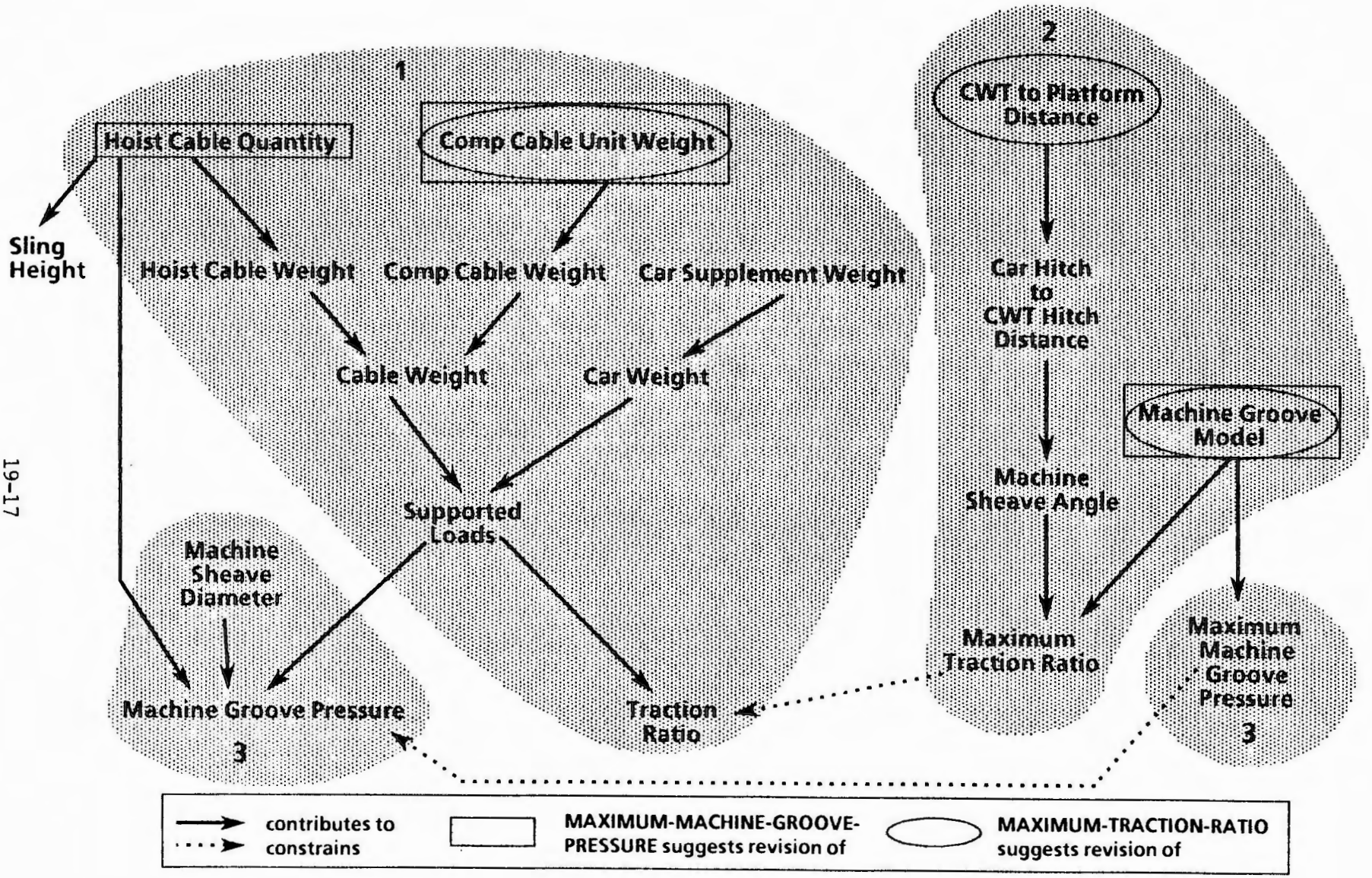


Figure 4: Subtask ordering to work on Maximum Traction Ratio before Maximum Machine Groove Pressure.

expert to indicate the important contributors (such as CAR-WEIGHT above) and limiting values (such as the tip-off that CAR-WEIGHT is "unusually high") that might serve as predictors of constraint violations, based on the expert's experience.

4.2 Start with the constraint with the most limited resources for satisfaction

Limited room to try to satisfy a constraint places a restriction on values modified to try to satisfy that constraint. It saves search time if those restrictions are discovered as soon as possible. That will prevent elaborating a plan based on values that will be abandoned later.

For example, recall that a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE calls for an increase of HOIST-CABLE-QUANTITY, a decrease of COMP-CABLE-UNIT-WEIGHT, and/or an upgrade of MACHINE-GROOVE-MODEL. If HOIST-CABLE-QUANTITY is at its maximum and COMP-CABLE-UNIT-WEIGHT is at its minimum, the only fix left for a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE is an upgrade of MACHINE-GROOVE-MODEL. So a violation of MAXIMUM-MACHINE-GROOVE-PRESSURE under these circumstances forces a limited choice on MACHINE-GROOVE-MODEL and it would be best to know this early on. Thus under these conditions, the plan shown in Figure 3 that starts with evaluating MAXIMUM-MACHINE-GROOVE-PRESSURE is a good idea.

Recall also that a violation of MAXIMUM-TRACTION-RATIO calls for an increase of COMP-CABLE-UNIT-WEIGHT, a decrease of CWT-TO-PLATFORM-DISTANCE and/or a downgrade of MACHINE-GROOVE-MODEL. If, on some other job, COMP-CABLE-UNIT-WEIGHT is close to its maximum and CWT-TO-PLATFORM-DISTANCE is close to its minimum, it would be better to start with MAXIMUM-TRACTION-RATIO as shown in Figure 4. In that case, a violation of MAXIMUM-TRACTION-RATIO might limit choices on MACHINE-GROOVE-MODEL in the other direction.

This procedure for ordering constraint evaluations and revisions is analogous to the high level plan selection that determines the order in which subproblems are addressed in the DSPL Mission Planning Assistant [Herman 86]. MPA selects a subproblem to start with based on where it expects resource limitations. For example, under ideal conditions for planning a mission, MPA will first consider selecting the best-suited aircraft and then figure out what base they can leave from.

Under limited-fuel conditions, when the destination is remote and no re-fueling is available, bases are narrowed down first and then aircraft are considered.

In the TRACTION-RATIO/GROOVE-PRESSURE example, the information that the plan selection is based on a consideration of what the relevant "resources" of each constraint evaluation subproblem is (i. e. , what are the possible revisions), how close they are to their limits and the direction in which the subproblem is likely to drive their values. These considerations establish how much play the problem solver has in trying to satisfy a constraint. Such information can be used by a compiler for automatically generating plan selection strategies. SALT could also describe design extension clusters and identify relevant resource dependencies and limitations to the expert. The expert could then use this information to craft strategies for ordering the work on the design subparts.

6. Conclusion

SALT adopts a representation scheme that requires users to build a fairly low level description of the domain-specific knowledge for a design task. Users build this representation in a bottom-up fashion with SALT's help in keeping track of how the pieces are fitting together. This places less burden on the memory and abstraction abilities of a single user than asking them to start with a top level description of ordered tasks. It is also useful in a situation in which a knowledge base must be built by multiple users, each expert in a different part of the problem. SALT's knowledge elicitation strategy is aimed at getting all of the knowledge needed for the expert system to produce good quality solutions. However, SALT's original compiler produced an expert system that didn't make use of some efficiency advantages of task clustering and ordering.

The present paper looked at some strategies for task structuring and ordering to improve the efficiency in SALT-generated propose-and-revise systems. The set of strategies was not meant to be exhaustive but to show the promise of the SALT representation language with respect to developing such strategies. While SALT does not explicitly represent subproblem groupings and orderings that might improve system efficiency, it routinely gathers much of the information and currently provides analyses that can aid in producing such organizations.

Acknowledgements

I would like to thank David Brown, Vasudevan Jagannathan and Ted Kitzmiller for their comments on an earlier draft of this paper. I am grateful to Bob Roche, first domain expert for VT, whose critique and feedback contributed to the SALT design. Please note that the examples in this paper are only loosely based on VT. The VT research was sponsored by Westinghouse Elevator Company, Randolph, New Jersey. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Westinghouse Elevator Company.

References

[Boose 84] Boose, J.
Personal construct theory and the transfer of human expertise.
In *Proceeding of the National Conference on Artificial Intelligence.*
Austin, Texas, 1984.

[Brown 85] Brown, D.
Failure handling in a design expert system. *Computer-Aided Design*, (17):436-441, 1985.

[Davis 77] Davis, R., B. Buchanan, and E. Shortliffe.
Production rules as a representation for a knowledge-based consultation program.
Artificial Intelligence (8):15-45, 1977.

[Dechter 85] Dechter, R. and J. Pearl.
The anatomy of easy problems: A constraint-satisfaction formulation.
In *Proceedings of the International Joint Conference on Artificial Intelligence,* Los Angeles, California, 1985.

[Dechter 87] Dechter, R. and J. Pearl.
The cycle-cutset method for improving search performance in AI applications.
In *Proceedings of the IEEE Conference on Artificial Intelligence Applications,* Orlando, Florida, 1987.

[Doyle 79] Doyle, J.
A truth maintenance system.
Artificial Intelligence (12):231-272, 1979.

[Eshelman 86] Eshelman, L. and J. McDermott.
MOLE: A knowledge acquisition tool that uses its head.
In *Proceedings of the National Conference on Artificial Intelligence.*
Philadelphia, Pennsylvania, 1986.

[Freuder 82] Freuder, E.
A sufficient condition for backtrack-free search.
Journal of the Association for Computing Machinery (29):24-32, 1982.

[Herman 86] Herman, D, J. Josephson, and R. Hartung.
Use of DSPL for the design of a Mission Planning Assistant.
Technical Report, Ohio State University, Department of Computer and Information Science, 1986.

[Kahn 85] Kahn, G., S. Nowlan and J. McDermott.
MORE: An intelligent knowledge acquisition tool.
In *Proceedings of the International Joint Conference on Artificial Intelligence,* Los Angeles, California, 1985.

[Klinker 87] Klinker, G., C. Boyd, S. Genetet, and J. McDermott.
A KNACK for knowledge acquisition.
In *Proceedings of the National Conference on Artificial Intelligence,* Seattle, Washington, 1987.

[Marcus 85] Marcus, S., J. McDermott, and T. Wang.
Knowledge acquisition for constructive systems.
In *Proceedings of the International Joint Conference on Artificial Intelligence,* pages 637-639, 1985.

[Marcus 86] Marcus, S. and J. McDermott.
SALT: A knowledge acquisition tool for propose-and-revise systems.
Technical Report, Carnegie-Mellon University, Department of Computer Science, 1986.

[Marcus 87] Marcus, S., J. Stout, and J. McDermott.
VT: An expert elevator designer that uses knowledge-based backtracking.
AI Magazine (8):41-58, 1987.

[Marcus 88] Marcus, S. and J. McDermott.
SALT: A knowledge acquisition language for propose-and-revise systems.
Artificial Intelligence, forthcoming, 1988.

[Mittal 86] Mittal, S. and A. Araya.
A knowledge-based framework for design.
In *Proceedings of the National Conference on Artificial Intelligence,* pages 856-865, 1986.

[Smith 86] Smith, S., M. Fox and P. Ow.
Constructing and maintaining detailed production plans: Investigations into the development of knowledge-based factory scheduling systems.
AI Magazine (7):45-60, 1986.

[Stefik 81a] Stefik, M.
Planning with constraints (MOLGEN: Part 1).
Artificial Intelligence (16):111-140, 1981.

[Stefik 81b] Stefik, M.
Planning and meta-planning (MOLGEN: Part 2).
Artificial Intelligence (16):141-170, 1981.

[van de Brug 86] van de Brug, A., J. Bachant, and J. McDermott.
The Taming of R1.
IEEE-Expert (1):33-42, 1986.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

USING THREE MINIMALLY BIASING ELICITATION TECHNIQUES FOR KNOWLEDGE ACQUISITION

Mary A. Meyer, Susan M. Mniszewski, and Alfred T. Peaslee, Jr.
Los Alamos National Laboratory, MS F600, Los Alamos, NM 87545

ABSTRACT
Bias in knowledge acquisition can hinder the building of systems that reflect the experts' actual heuristics and operational procedures. Bias is defined, here, as an altering or misrepresentation of the expert's thought processes. This paper focuses on three elicitation techniques--the verbal protocol, a type of verbal probe, and the ethnographic query. These techniques were tailored to avoid the introduction of bias and to be easy for the lay person to use. For each technique, detailed guidance is given on situations in which it can be used, how to set up for its use, and how to administer it.

INTRODUCTION

This paper provides guidance on how to use three simple elicitation techniques--the verbal protocol, a type of verbal probe, and the ethnographic query. These techniques were selected and tailored to avoid introducing bias. Bias is defined here as an altering of either the expert's thought processes or the reporting of these. This definition assumes that the expert has a natural way of solving a problem that can be changed or misrepresented by the attempt to discover it. This natural method can be changed if the expert alters his/her way of solving the problem as a result of information received during the elicitation process. It can be misrepresented if experts fail to give an accurate accounting of their thoughts or if the knowledge engineer misinterprets these thoughts. Thus, bias can enter into knowledge acquisition from two sources, the expert and the knowledge engineer.

Experts may alter the way that they solve problems or their reporting of this process for several reasons. First, most people have an emotional need to be accepted and to receive approval (Zimbargo, 1983; Meyer, 1984). Second, people are generally unaware of how they make decisions (Hogarth, 1980:ix), such as in solving problems. Yet, as Denning (1986:345) notes, Western scientific tradition assumes that the problem-solving process can be precisely stated, so it is difficult for experts to acknowledge their ignorance. Thus, in elicitation situations, experts are likely to be extremely responsive to what they believe the knowledge engineer wishes to hear. The knowledge engineer's expectations or wishes concerning problem solution are communicated, often without awareness, through questions, responses to the expert, and body language. The experts are likely to acquiesce unconsciously to suggestions of how they might have solved the problem or to change their manner of solving future problems.

The second source of bias stems from the knowledge engineer's interpretation of the expert's data. It is human tendency to perceive and interpret incoming information in a selective way that supports what is already

believed or known. Sometimes, this leads to a misinterpretation, or biasing, of the information. For example, when the expert mentions a new term, we, as knowledge enginers, tacitly assume a meaning based on our experience with similar sounding words.

These two sources of bias can interfere with knowledge acquisition, with building a system that reflects the "expert's actual heuristics and operating procedures" (Waterman, 1986:161). Bias from the expert or the knowledge engineer can confuse and slow knowledge acquisition. For example, if the knowledge engineer asks the expert whether x, y, and z have been considered, the expert may answer "yes" or even begin doing so on future problems. If, however, the expert has not previously considered these factors, discrepancies are likely to arise between the expert's claim and his/her solutions or between the expert's previous and future solutions. Another instance of bias occurs when the knowledge engineer is interpreting the expert's knowledge and mentally building his/her own, slightly different, knowledge structure. As the knowledge engineer receives new knowledge from the expert, inconsistencies and the question of what is causing them must be faced. Thus, the discrepancies between what is reported by the expert or interpreted by the knowledge engineer and the expert's actual thinking can complicate the process of acquiring knowledge.

There has been little literature on the problems of bias in knowledge acquisition, with the exception of Cleaves (1986). Most of the research in the area of bias has been done in the fields of decision analysis or judgment theory. These fields have traditionally viewed bias in a different light than artificial intelligence. They have regarded bias as a skewing from what the expert's answer should be, given statistical norms and logical rules, as opposed to what it is (Henrion and Cooley, 1987; Mumpower et al, 1987). For this reason, some of the approaches to countering bias that emerge from these other fields may not be directly applicable to artificial intelligence.

There has been a more general lack of literature on exactly how to extract the expert's problem-solving processes. Although the difficulty of knowledge acquisition has been noted frequently in the literature (Cleaves, 1986; Belkin et al, 1986; Hoffman, 1987; Williamson, 1987), there have been few sources on how exactly to perform this task (Hoffman, 1987:53). Currently, there are about four methods of knowledge acquisition: (1) observing experts at work in interaction with someone, (2) structuring the task or problem in different ways to obtain their solution, (3) obtaining their verbal descriptions of their thoughts while they solve problems, and (4) questioning the experts. While there are beginning to be sources on both observations (Belkin et al, 1986) and task structurings (Hoffman, 1987), there is still nothing, to our knowledge, on the last two methods. Items (3) and (4) can be classified as elicitation techniques because they are designed to evoke verbal responses from the expert.

Several reasons exist for there being little knowledge acquisition literature on elicitation techniques. First, the way in which these techniques are learned does not lend itself to publication. Traditionally, elicitation techniques in anthropology, psychology, and sociology have been taught in laboratory situations and learned by doing. Thus, even within these specialized fields, there are few sources on elicitation. Second, the existing literature is specialized for a particular discipline or situation

and is not easily generalizable to others. Third, there is difficulty in communicating elicitation techniques through a written medium because these techniques involve responding to differing levels of information being simultaneously and nonverbally communicated. Last, most of the mechanics of elicitation become automatic to the experienced practitioner. Thus methodological tidbits, such as where you should sit in relationship to the expert and the reasons for this position, become relatively inaccessible for retrieval from the practitioner's mind. It is ironic but fitting that obtaining information on how to perform elicitation meets with the same basic obstacle as confronting knowledge acquisition in general.

The lack of how-to information is a serious one because many knowledge engineers could be considered lay persons in elicitation since frequently their backgrounds are in computer science, engineering, or other "hard sciences." For this reason, many of them would benefit from detailed descriptions of how and when to use elicitation techniques.

The lack of literature on how to use elicitation techniques, especially with regard to bias, was the stimulus behind this paper.

THE THREE ELICITATION TECHNIQUES

The three elicitation techniques--verbal protocol, a type of verbal probe, and the ethnographic query--were selected because they were not inherently biasing and could be adapted to further reduce the possibility of bias. These techniques are unobtrusive and therefore less likely to lead experts to change or misrepresent their thinking. They are easily put in place at the location where the expert normally solves problems and do not require special settings or equipment. Even subtle changes in the setting can cascade and alter the expert's natural means of problem solving. These three techniques do not demand that the knowledge engineer constantly intrude on the expert's thoughts by asking questions, thus altering the expert's normal thought progression. These techniques do not provide the experts with much information on the knowledge engineer's views and are less likely to lead them to alter their thinking processes, or descriptions of these. In addition, these techniques have the advantage of focusing the knowledge engineer's attention on the expert's thinking (e.g., by requiring that the knowledge engineer first listen without asking questions and then use the expert's own words in phrasing questions). This focus helps counter the natural tendency of the knowledge engineer to recast the expert's thoughts into his/her own.

As mentioned above, these techniques have been adapted by the authors to further minimize opportunities for bias. For example, the seating and note-taking logistics of the verbal protocol have been developed to keep knowledge engineers in the background where they are less likely to influence the expert. The verbal probe has been tailored through the selection of nonleading question phrasings, and the ethnographic query timed so that it does not upset and alter the expert's train of thoughts.

The three techniques were also selected because they had the potential of being easy for a lay person to use--most knowledge engineers lack backgrounds in this area. These techniques are simple and can be broadly applied. They can be combined with each other and with other knowledge acquisition methods to extract additional or more detailed information. For

example, the verbal probe can be used with the observational and task-structuring methods to examine why the expert responded as he/she did. The verbal probe can be used immediately after the expert finishes the problem, and the ethnographic technique can be used to pursue questions arising from the expert's reply to the probe.

These techniques have been refined through several applications to make them easier to use with respect to reducing bias. One or more of the techniques have been used on studies of the following:

1. Magnetic fusion scientists projecting the achievement of milestones in the U.S. magnetic fusion program (Meyer et al, 1981).
2. Weapons planners forecasting U.S. needs for the year 2000 (Meyer et al, 1982).
3. Statisticians solving walk-in consulting types of problems (Booker and Meyer, 1988).
4. Nuclear engineers evaluating the performance of their computer code in simulating selected reactor phenomena (Meyer and Booker, 1987).
5. Military personnel determining tank maneuvering tactics for selected defense scenarios (Meyer, 1987).

This paper will proceed from the simplest technique to those more suited to the gathering of detailed information. How to combine these techniques with others will also be discussed. The following information will be provided on each technique: potential uses; advantages and disadvantages, especially with respect to bias; suggested setup procedures; illustrations of its use; and ideas on how to administer it. The illustrations were developed as part of a seminar for students at the Knowledge Systems Laboratory at Los Alamos National Laboratory. The domain of horse training, specifically, determining a feed program for a horse, was selected for the seminar because the person available for the demonstration had expertise in this area and because the area was judged to be easily comprehendible.

Verbal Protocol

The verbal protocol technique is sometimes referred to as "thinking aloud" because the experts are instructed to think aloud as they progress through the problem. This technique has its origins in psychology (Ericsson and Simon, 1980). The objective of this technique is to have experts give first priority to working through the problem and to simply voice their thoughts as a flow-of-consciousness. Thus, the focus of the expert should be on the problem rather than on the knowledge engineer or on verbalizing. In addition, the expert should work through the problem naturally and at normal speed. Working through the problem more slowly would not only be difficult for the expert (Prerau, 1987:47) but could alter either the problem-solving process or solution.

The verbal protocol is thought to have two advantages: (1) it obtains the experts' verbal reports of problem-solving processes that are in their short-term memory, and (2) it does not require questioning by the knowledge engineer and thus is not as likely as other methods to introduce bias from this source. Because the knowledge engineer is not asking questions, there is less opportunity to give, consciously or unconsciously, leading information to the expert. As mentioned earlier, experts are often influenced by what they perceive to be the knowledge engineer's preferences or interests. In addition, if the experts are simply voicing their thoughts

and not translating their jargon for the knowledge engineer's understanding, the experts are less prone to oversimplify or give otherwise inaccurate descriptions.

There are two significant disadvantages of the verbal protocol. One disadvantage is that it does not usually provide as much information as the knowledge engineer desires. Although this could be said of all knowledge acquisition methods, the verbal protocol places the knowledge engineer in the passive role of simply accepting what the expert provides. In particular, if the expert is progressing through an especially difficult portion of the problem, the thinking aloud is likely to diminish or stop. The second disadvantage is, as Hoffman notes (1987:55), that not every expert is capable of using verbal protocol. In our experience, about one expert in thirty is totally unable to use this technique. Verbalizing their thoughts seems to conflict with their thinking and cause them extreme frustration. The majority of experts have varying proficiencies in the use of verbal protocol and become more articulate as they become accustomed to it.

Of the disadvantages of the verbal protocol, the first is the easiest to overcome. The verbal protocol can be combined with one or both of the other elicitation techniques mentioned in this paper to draw out more information. The second disadvantage can be addressed by testing potential experts on a sample problem. If the verbal protocol is to be a major means of knowledge acquisition for the project, those who are most adept at using the technique can be selected. If however, the knowledge engineer has been provided with only one or two experts, neither of whom is comfortable with the verbal protocol, it is best to learn this early in the project. Then, the knowledge engineer can use other knowledge acquisition methods.

Uses of the Verbal Protocol. Generally, the verbal protocol is useful for providing the knowledge engineer with an introduction to how the expert works through a problem, what information is needed, and what jargon is used. If textual information has been available, using verbal protocol can be a convenient next step toward gaining background in the domain.

Verbal protocol can be used to elicit general information on how multiple experts work through the problem; for instance, verbal protocol can be used with several experts to determine which expert's problem solving should be investigated in greater depth. It can also be used in combination with task structuring to learn how the same expert solves different problems. For example, a few parameters of the problem could be varied to learn if they have a noticeable effect on the expert's way of thinking.

The verbal protocol technique is also helpful in allowing the knowledge engineer to detect inconsistencies in the expert's reasoning. Often, the expert's interpretation of a problem will change as he/she works through the problem. Experts are likely to make some assumptions in interpreting the question, especially if they feel they are missing some information that they need to proceed. It is also common for them to subtly and unconsciously modify the structure of the problem (e.g., its definitions and boundary conditions). The verbal protocol allows the knowledge engineer to trace the evolution, and sometimes convolution, of the expert's thinking through time.

Although the verbal protocol has many uses, there are several restrictions to its use. First, the verbal protocol should only be used in face-to-face situations. Many of the expert's verbalizations are short phrases given within the context of the problem and would not be comprehensible if the knowledge engineer were not physically present. In addition, the verbal protocol technique should only be used on one expert at a time. Following an expert's verbal reports requires the knowledge engineer's full attention. Also, it is easier for the expert to focus on solving the problem if other experts are not present. Other experts can distract, influence, or otherwise bias the expert.

The verbal protocol can also be used with either or both of the other elicitation techniques. For example, if the verbal protocol is used while the expert is working on the problem, the verbal probe can be used immediately after the expert reaches a solution. The ethnographic technique can be employed. then, to pursue questioning on the expert's replies to the verbal probe. This convenient combination of the techniques will be illustrated in later sections.

How to Set Up for the Verbal Protocol. Place your chair slightly behind the expert and to the opposite side of his/her handedness (i.e., for a right-handed expert, you would want to be to the expert's left). There are several advantages to this positioning. First, the expert will not be as able to see what or when you are taking notes. If the expert becomes aware of what you are interested in, this awareness may influenced, or bias, the response. Second, these positions allow you to see, unobtrusively, what the expert is looking at, marking, or writing.

Design the problems, at least initially, so that they are like those in which the expert has expertise. Otherwise, there is little point in using the expert. The elicitation sessions should be set up so they resemble, as much as possible, the environment in which the expert usually solves such problems. Otherwise, you may be introducing factors that change the expert's usual way of thinking. The sessions can be conducted in the expert's customary work place, if interruptions can be controlled. For instance, telephone calls can be handled by call forwarding.

Obtain copies of whatever visual aids (e.g., tables, graphs, and equations) or references the expert will be using. This practice allows you to follow what the expert is viewing even when you can not see it over his/her shoulder because of the distance and the print size. It also provides a hard copy for recording what the expert is looking at and marking. Hard copies should also be obtained if the expert is solving problems on a computer. In the later case, the hard copies are of the computer screens.

Take notes rather than rely on the expert or recording devices. Experts are unreliable in taking notes, and should not try to provide you with detailed written accounts because this activity is likely to be done at the expense of their thinking. Thus, not only do you obtain little data but it is likely to be biased as well. Recording devices, by themselves, do not provide complete records. For example, they do not show the marks that experts make on their papers as they think. In addition, recording devices malfunction, so a backup copy, yours, is likely to be needed.

Emphasize that the expert is to work through the problem rather than to talk hypothetically about how it could be solved. Experts are often unaware of, or mistaken about, how they actually solve a problem. They, therefore, provide more reliable (less biased) information if they are verbalizing while they are solving the problem. The message on actually working the problem need only be delivered the first couple of times that you work with the expert.

Emphasize the importance of thinking aloud when instructing the expert to begin solving the problem. The first time an expert solves a problem in your presence may remind him/her of a test situation in school. As a result, the tendency may be to rush through the problem to give you the solution. You must contrive to convince the expert to think aloud without emphasizing the importance of verbalizing at the expense of problem solving. The latter alters the way the expert would normally work the problem and is, therefore, undesirable.

Illustration of Verbal Protocol. The knowledge engineer in this example wishes to learn, in general, how the expert plans a feed program, what information on the horse and on the types of feed are needed, and what concepts (e.g., the horse's metabolism) are considered.

The expert is presented with a written copy of the problem: "What feed program would you start this colt on? The colt is 6 months old, 550 lbs., has an average metabolism, and will receive light exercise through ponying. Assume that the colt has normal good health."

Knowledge engineer--Solve this problem as you do others you receive in this field. Please try to think aloud as you work your way through the problem. Your thinking aloud is as important to me as the answer that you reach.

Expert--(The expert slowly reads the question aloud.) The first thing that I'll need to find out is what this colt would weigh full grown. (The expert scans some xeroxes that were previously copied from reference books.) Let's see, at six months and 550 lbs., he will be about 1100 to 1300 lbs. full grown.

Next, I need to find the balance between hay and grain that I'd feed a horse of this age. (The expert looks at another table.) ...Four lbs. of hay and 8 lbs. of grain. However, I like to feed more hay so I will aim for 6 lbs. of hay and 7 lbs. of grain.

Other constraints that I have are the amount of protein in lbs. needed for a colt that will mature to a 1100- to 1300-lb. horse. (The expert refers to another table.) I'll want to balance between 1.74 and 1.89 lbs. of protein per day.

(The expert examines another chart.) I also need to balance protein in the overall diet to about... 14.5%, calcium to... 6% and phosphorus to... 0.45%.

Foods that I like to feed in the diet are oats, corn, and soybean meal. I also like a feed supplement for extra protein, Horsecharge, and alfalfa and timothy hay. I'm getting their percentages of protein,

calcium, and phosphorus from the charts. (The expert ceases thinking aloud, records the percentages of these feeds, and begins using a calculator.)

(The expert writes a list of feeds and weights and leans back in the chair, signaling that the exercise has finished.)

For the purpose of this illustration, the expert will be frozen in time at this stage of having just reached an answer.

How to Conduct the Verbal Protocol. Frequently, the expert will stop thinking aloud and need prompting to resume. Several types of prompts are given below. One reason for using different prompts is to avoid repetitiveness that can be irritating to both expert and knowledge engineer. Another reason is to use the most subtle prompt possible to remind, rather than to distract, the expert. Distraction can cause experts to lose their place and then solve the problem differently than they would have otherwise (i.e., bias). The prompts are given in order of most to least subtle.

1. Look intently at the expert and lean forward slightly.
2. Look at the expert, lean forward, and clear your throat.
3. State "please try to think aloud."

The first prompt would be used with an expert who only needs the occasional reminder to think aloud. The expert perceives the knowledge engineer's forward movement almost subliminally and then responds. The second prompt is slightly stronger because it triggers associations concerning the expert's throat (e.g., my throat should also be making sounds). The third prompt would be used on the expert who failed to notice or respond to the first two prompts.

Generally if the expert is counting or otherwise performing calculations, a reminder to think aloud would be inappropriate. The expert could lose his/her place. If it is necessary to prompt the expert at this time, the first prompt would be better than the third. The nonverbal aspect of the first prompt allows the expert to note it without being distracted.

A Type of Verbal Probe

The verbal probe is questioning done at a particular time and in a specific manner. Like verbal protocol, verbal probes are borrowed from psychology (Ericsson and Simon, 1980). However, with the verbal probe, unlike the verbal protocol, the knowledge engineer asks questions rather than only encouraging the expert to speak. The term "verbal probe" can refer to different types of probes but in this paper will only refer to a probe that is indirectly phrased, concurrently timed, and specifically focused (Meyer, 1987). A probe with these traits was selected because it was judged more likely to produce reliable reports and less likely to introduce bias. Indirect phrasing such as "Why did you give that answer?" is less likely to influence the expert than one that suggests options (e.g., "Did you give that answer for x, y, or z reasons?"). In addition, the response to the indirect phrasing provides better evidence of the expert's thoughts. The probe is concurrently timed, which means that the questions are asked while the expert retains the problem-solving processes in short-term memory. With this timing, the expert is less likely to resort to hypothesizing how he/she might have solved the problem (i.e., as in plausible reasoning). Lastly,

the probe is focused on a specific problem rather than on a hypothetical problem or several problems. The intent is to gather data on how the problem was solved rather than on the expert's vague generalizations or speculations.

Uses of the Verbal Probe. The verbal probe can be used to obtain general (coarse granularity) information on one or more expert's problem solving. In individual, face-to-face interviews, the verbal probe can be used after the verbal protocol. In settings where multiple experts are solving problems, the verbal probe can be used, by itself, to survey the experts' reasoning.

The verbal probe can also deliver detailed (fine granularity) data if it is used with the ethnographic method in an individual, face-to-face interview.

How to Set Up for the Verbal Probe. If you are using the verbal probe in an individual, face-to-face interview, set up as mentioned earlier for the verbal protocol. As a general rule, individual interviews are used for pursuing more detailed information than can be obtained in group settings.

If there will be multiple experts in a group setting, have the group moderator mention, in advance, that the experts will be asked to provide their reasons for giving the answers that they did. The experts should be asked to provide this information verbally because people, in general, do not provide good written records of their reasoning. The records are usually sketchy and, for this reason, more likely to be misinterpreted by the knowledge engineer than their more complete verbal counterparts.

Illustration of the Verbal Probe. Assume that the expert has used verbal protocol, as previously illustrated, and is at the point in time where the problem has just been solved. In this example, the expert will be asked for an answer and then administered the verbal probe to learn why that answer was given.

Knowledge engineer--What was your answer?
Expert--The diet, per day, that I would recommend for this horse is 4 lbs. of oats, 1.5 lbs. of corn, 0.5 lbs. of soybean meal, 1 lb. of Horsecharge, 4 lbs. of alfalfa, and 2 lbs. of timothy. This is, of course, only the starting point.
Knowledge engineer--Why did you give that answer--that feed program?
Expert--Well, it provides the right amounts of protein, calcium, and phosphorus for a horse to grow at this age.

The last response constitutes the expert's answer to the verbal probe. Again, the expert will be frozen in time, while the discussion continues.

How to Conduct the Verbal Probe. Ensure that the verbal probe is worded in the past tense (e.g., "Why **did** you give this answer?") to emphasize that you wish to know how the expert actually solved the problem. Some experts will begin, even after solving a problem, to tell you how they or a colleague **could** have solved it. (Your response to such a beginning should be something like: "I am interested in **your** thinking and how **you** solved this problem.")

Check for tautological responses. Tautologies are reasons that do not truly provide the "why" information that you are seeking. For example, "I gave that feed program because it seemed right" is a tautological response. Although, the tautology here is obvious, tautologies can be difficult to discern when you have been immersed in an unfamiliar domain, vocabulary, and/or are tired. Frequently, you can obtain the desired information by using the ethnographic technique to probe on the tautology.

Ethnographic Technique

The ethnographic method involves restating or inflecting the expert's words to make them into questions. For example, if the expert mentioned something about "light exercise through ponying," the knowledge engineer could use the ethnographic query, "ponying?" This simple technique is from ethnology (cultural anthropology) and is used to pursue elicitation in greater depth.

The advantages of this technique regarding bias are twofold. First, the questions of the knowledge engineer provide little information that could influence the expert's thinking. This approach is successful because the expert's own words are used for the questioning. Second, this technique acts as a check on the knowledge engineer's tendency to fill in the expert's thinking with his/her own. Use of this technique forces the knowledge engineer to obtain the expert's definitions. For example, in the following illustration of the ethnographic query, the expert uses a phrase, "keeping his weight on," which the knowledge engineer would have interpreted in a different way than the expert. (The knowledge engineer had first assumed that this meant not loosing weight.) However, use of the ethnographic technique enabled the knowledge engineer to learn immediately that this phrase meant gaining weight at the desired rate.

Uses of the Ethnographic Technique. The ethnographic technique is generally useful for probing to a detailed level of knowledge without biasing the expert's thoughts. It can be used in problem-solving sessions or discussions with the expert. For example, the ethnographic technique can be used in discussion sessions to pursue previously mentioned terms or concepts. Another possibility is to let portions of the expert's verbal protocol be the stimulus for the questioning. Prerau (1987:47) suggests playing short sections of the audio tape of the expert's problem-solving session.

In general, the ethnographic technique should not be used in a group setting because it is an intensive technique and, therefore, best suited to one-on-one situations. However, it can be employed in group settings if it is only used briefly to obtain experts' definitions. For example, if a group of experts were demonstrating how they work through a problem and mentioned some term, the ethnographic technique could be used to extract the expert's definition of that term.

How to Set Up for the Ethnographic Technique. How the ethnographic technique is set up is usually determined by the setup for the other techniques used or by the situation in which it is used alone. For example, if it were used with the verbal protocol, the verbal protocol's setup would be used. If the ethnographic method is to be used by itself, no special setup would be needed. For example, if it is to be used on one expert at a

time, it can be set up as for a conversation (e.g., you would sit adjacent to or across from the expert).

Illustration of the Ethnographic Technique. The expert has just answered the verbal probe explaining why that answer was given. The ethnographic technique could be used to investigate this information or information given earlier. Earlier, the expert had given the feed program and added, "Of course, this is only the starting point." For the purpose of this illustration, the ethnographic technique will be applied to the reply to the previous verbal probe and then to the subject as it develops.

Knowledge engineer--A starting point?
Expert--Yes, because the rations may need to be adjusted, if the colt is getting fat or if he's not keeping his weight on.
Knowledge engineer--Not keeping his weight on?
Expert--Yes, not gaining as he should at this time.
Knowledge engineer--What do you mean by "at this time"?
Expert--At his age, 6 months, given his expected total weight, he should be gaining between 1.5 to 2 lbs. per day.

The ethnographic technique was used to follow a line of the expert's thoughts to a detailed level where an operational definition of "not keeping his weight on" was obtained. The knowledge engineer stepped down three levels in questioning, as shown in the figure. The ethnographic technique is effective at taking the questioning to the desired level of granularity, in this case to a definition that would be represented and programmed. In another case, the ethnographic technique could be used to branch across several topics at a more general level.

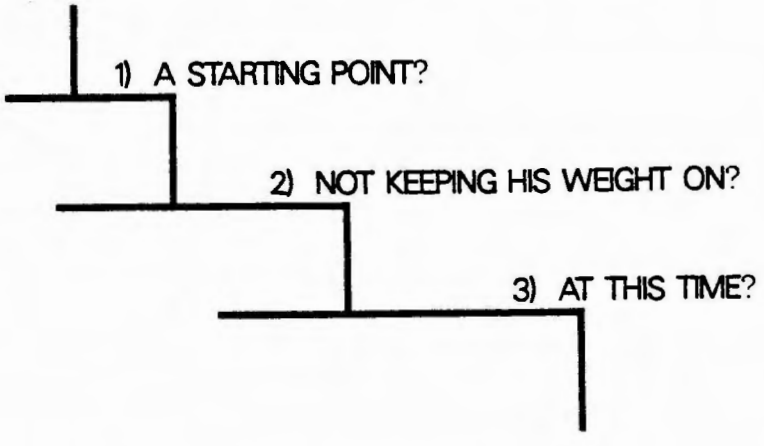


Figure 1. Series of ethnographic questions.

How to Conduct the Ethnographic Technique. Vary the length of the ethnographic questions so that they do not become tedious. For example, if the expert said, "The books give samples and I start with these," a short ethnographic query would be, "samples?" For variety, a longer query would be, "What samples do the books give?"

If the ethnographic technique is used in problem-solving sessions, plan to use it only after the expert has reached the point in problem solving that

satisfies your goals. Otherwise, the expert is likely to lose what he/she has in short-term memory while responding to your questioning. Then, if you resume questioning where the interruption occurred, the expert will have to guess at his/her previous thoughts. Bias can result. For this reason, the ethnographic technique is best used after other techniques or by itself. Occasionally, the ethnographic technique can be inserted during the use of another technique but only if it is asked quickly and not pursued to fine granularity.

RECOMMENDATIONS

The three elicitation techniques suited to knowledge acquisition described in this paper are recommended for use singly, in combination with each other, or with other knowledge acquisition methods. Although these techniques are broadly applicable, it is recognized that they may not fit every project.

Knowledge engineers need to be sensitive to the existence of bias, which complicates the already difficult task of knowledge acquisition. Whether the goal of knowledge acquisition is capturing the expert's performance or improving on it (Henrion and Cooley, 1987), the immediate focus must be to extract the knowledge that the expert actually uses in solving problems. To this end, more literature is needed on knowledge acquisition methods, particularly on how to avoid bias.

ACKNOWLEDGEMENTS

The interest and support of the Knowledge Systems Laboratory at Los Alamos National Laboratory led to the seminar from which this paper was written. The material presented had been developed over time through several studies. These studies would not have been possible without the information provided by individuals in the Statistics, Reactor Safety, Military Systems, Simulations and Software, and Systems Analysis groups of Los Alamos National Laboratory. The financial support of the Nuclear Regulatory Commission (FIN #A7225-5) for the latest of these studies is most gratefully acknowledged.

REFERENCES

Belkin, N.J., Brooks, H.M., and Daniels, P.J. (1986). Knowledge Elicitation Using Discourse Analysis, in Proceedings from the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (Boeing Computer Services, Seattle, WA), pp. 3-0 to 3-16.

Booker, J.M., and Meyer, M.A. (1988). Sources and Effects of Interexpert Correlation: An Empirical Study. IEEE Systems, Man, and Cybernetics, Vol. 18, No. 1, pp. 135-142.

Cleaves, D.A. (1986). Cognitive Bias and Corrective Techniques: Proposals for Improving Elicitation Procedures for Knowledge-Based Systems, in Proceedings from the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada (Boeing Computer Services, Seattle, WA), pp. 9-0 to 9-11.

Denning, P.J. (1986). The Science of Computing: Will Machines Ever Think? American Scientist, Vol. 74, pp. 344-346.

Ericsson, K.A., and Simon,H. A. (1980). Verbal Reports as Data, Psychological Review, Vol. 87, No. 3, pp. 215-250.

Henrion, M., and Cooley, D.R. (1987). An Experimental Comparison of Knowledge Engineering for Expert Systems and for Decision Analysis, in the Proceedings of the American Association for Artificial Intelligence, Los Altos, California: Morgan Kaufman Publishers. Forthcoming.

Hoffman, R.R. (1987). The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology, AI Magazine, Summer, pp. 53-66.

Hogarth, R.M. (1980). Judgement and Choice, New York: John Wiley and Sons.

Meyer, M.A., Booker, J.M., Cullingford, H.S., and Peaslee, A.T., Jr. (1981). A Data-Gathering Method for Use in Modeling Energy Research, Development and Demonstration Programs, in T. N. Veziroglu, (Ed.), Energy Programs, Policy, and Economics: Alternative Energy Sources, Florida: Butterworth Publishers, pp. 421-430.

Meyer, M.A., and Peaslee, A.T. Jr., and Booker, J.M. (1982). Group Consensus Methods and Results, Energy Systems Group, Los Alamos National Laboratory technical report LA-9584-MS.

Meyer, M.A. (1984). Human Factors Affecting Subjective Judgments, in the Proceedings of the Thirty-First Conference on the Design of Experiments in Army Research Development and Testing, 1986, pp. 85-98.

Meyer, M.A., and Booker, J.M. (1987). Sources of Correlation Between Experts: Empirical Results from Two Extremes, Los Alamos National Laboratory technical report LA-10918-MS and NUREG/CR-4814.

Meyer, M.A. (1987). Eliciting Subject's Problem-Solving Processes on Computerized Exercises: A Cognitive Interview, Los Alamos National Laboratory, Statistics Group internal report A-1/87-U-601.

Mumpower, J.L, Phillips, L.D., Renn, O., and Uppuluri, V.R.R. (1987). Expert Judgment and Expert Systems, NATO ASI Series, New York: Springer-Verlag.

Prerau, D.S. (1987). Knowledge Acquisition in the Development of a Large Expert System, AI Magazine, Summer, pp. 43-51.

Waterman, D.A. (1986). A Guide to Expert Systems, Reading, Massachusetts: Addison-Wesley.

Williamson, M. (1987). Breaking the Expert Systems Bottleneck, Computers in Science, November/December, pp. 62-65.

Zimbargo, P.G. (1983). To Control a Mind, Standford Magazine, Winter, pp. 59-64.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

A Methodology and Tool for Knowledge Acquisition in KEATS-2

Enrico Motta, Tim Rajan, and Marc Eisenstadt

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

Abstract: In this paper we present our view of the stages involved in knowledge acquisition, stressing the need for tools supporting empirical data analysis as well as allowing a smooth transition from data analysis to domain conceptualization and finally to the system implementation. The strengths and weaknesses of our earlier system, KEATS-1, have led us to embark upon the design and implementation of a new system, KEATS-2, which provides a more coherent and integrated framework for performing both bottom-up and top-down knowledge acquisition. This paper introduces the first of our redesigned KEATS-2 modules, entitled Acquist, and describes how it provides support for data analysis and domain conceptualization.

Acknowledgement: This research is supported by a grant from British Telecommunications, plc. John Domingue participated in the discussions which led to this paper. Steven Rose and Mike Stewart of the Open University's Brain Research Group provided valuable domain expertise.

1. THE PROBLEM OF KNOWLEDGE ACQUISITION

The most popular principle in knowledge based systems states that the performance of an expert system critically depends on the amount of knowledge embedded in the system (Feigenbaum, 1977). Therefore the knowledge engineer usually spends a great deal of time eliciting knowledge from domain experts and even more trying to make sense of the data acquired. This combined activity of eliciting, interpreting and organizing the knowledge acquired from the expert is called 'knowledge acquisition', and is often described as a lengthy and painful process. A number of factors related to the problem being tackled, the techniques used for knowledge elicitation and the personality of the domain expert can make this process extremely difficult. In fact, problems can arise, due to the complexity of the domain and to possible mismatches between the knowledge elicitation technique and the structure of the problem (e.g. techniques such as protocol analysis work very badly for domains which are best represented declaratively). Moreover, recent empirical studies have shown that the efficacy of a particular knowledge elicitation technique can also be affected by the personality traits of the domain expert (Burton et al., 1987).

Much research has tackled the problem of devising tools (Anjewierden, 1987) and techniques (Olson and Reuter, 1987) that could support, speed up and eventually automate the knowledge acquisition process. Although all of these proposals attempt to tackle the same problem, they vary a great deal depending on the approaches, functionalities, underlying models and assumptions. In this paper we address some of the issues concerning knowledge acquisition, trying to clarify the nature of the knowledge acquistion process, and to point to the kind of software support the knowledge engineer can be provided with.

The rest of the paper is organized as follows. In section 2 we outline a functional model of knowledge engineering that will provide both the basis for clarifying the typology of the knowledge acquisition task and the theoretical framework for devising a knowledge acquisition tool. In section 3, we describe how these considerations led to the implementation of data analysis and domain conceptualization tools in the original KEATS system (Motta et al., 1988), hereafter referred to as KEATS-1. The strengths and weaknesses of KEATS-1 have in turn led us to the development of a new system, Acquist, that is the kernel of the knowledge acquisition facilities being provided as part of our ongoing KEATS-2 project. Section 4 describes Acquist's structure, and its support for bottom-up

and top-down approaches to knowledge acquisition. Finally, section 5 provides conclusions and a restatement of our current view of the state of knowledge acquisition.

2. A STAGE-ORIENTED MODEL OF KNOWLEDGE ACQUISITION

2.1 Critical stages

Figure 1 sketches a functional model of knowledge engineering where the ovals represent activities and the rectangular boxes inputs and outputs. The first three ovals, starting from the top, Knowledge Elicitation, Data Analysis and Domain Conceptualization, are the activities that define the knowledge acquisition stage. Hence, consistently with other descriptions (Hayes-Roth et al., 1983), our model represents knowledge engineering as consisting of the acquisition, representation, implementation and debugging of (a model of) the expert reasoning and phenomenology for a chosen target domain. Knowledge acquisition is in turn defined as a composite activity that includes both the elicitation and the interpretation of the data. We want to stress this last point, as knowledge acquisition and knowledge elicitation are often confused. For instance, the paper by Olson and Reuter claims to provide a survey of knowledge acquisition techniques, while almost all of the mentioned techniques only address the *elicitation* level of the task.

The bold arrows represent the ordering relations between the different subtasks. For instance, knowledge elicitation is meant to produce the 'raw data' that are the input for the data analysis task. As the interactions between the various tasks can be fairly unpredictable, the knowledge engineer can be forced to backtrack to a previous activity to gather new knowledge, modify existing data, or detect errors and misconceptions. The right-angled arrows emanating from the left hand side of each oval indicate the iterative nature of (and inherent interdependencies among) all of these activities.

We could, of course, further characterize this model, by opening up the boxes and the ovals and analyzing the subtasks that exist at a more detailed level. We have only hinted at this by drawing a dotted line down the centre of every rectangle in order to indicate potentially divergent foci of interest. The two most obvious foci are (a) what the expert does (left hand side of each rectangle), and (b) how objects in the domain behave (right hand side of each rectangle). Another possible source of expertise is the end-user, who can provide the knowledge engineer with knowledge about the expected *modality* of the system (Wielinga and Breuker, 1986). The two large braces encompassing the top and bottom half of the diagram, respectively, indicate a concomitant emphasis (by the knowledge engineer) on (a) *overt behaviour*, as typified by 'what happens' phenomenologically to objects within the domain as well as by the expert's performance within it, and (b) *internal representations* sufficient to perform the desired task to the satisfaction of the knowledge engineer, domain expert, and end user.

Finally, the reader should bear in mind that, although the model is complete in representing the sequence starting with the knowledge elicitation sessions and ending with the system's implementation, there are a number of tasks that the knowledge engineer has to perform which are not covered here. These tasks include some preliminary activities, such as domain and problem assessment, and post-mortem activities such as system maintenance. As these problems are tangential to the problem of knowledge acquisition, they will not be included in our discussion. We note, however, that domain assessment, typically regarded as a 'feasibility' exercise, is itself similar to the activities shown in Figure 1 up to and including domain conceptualization, and therefore follows a comparable course.

The functional characterization of knowledge engineering provides a methodological basis for understanding and driving the work of the knowledge engineer. Moreover, on the practical side, we can use the task decomposition sketched above as a guideline for designing tools that could support the knowledge engineer in building expert systems. This idea has led to the development of the first and the second version of the KEATS system, which will be described in sections 3 and 4. In the

remainder of this section we concentrate upon describing the stages that make up knowledge acquisition.

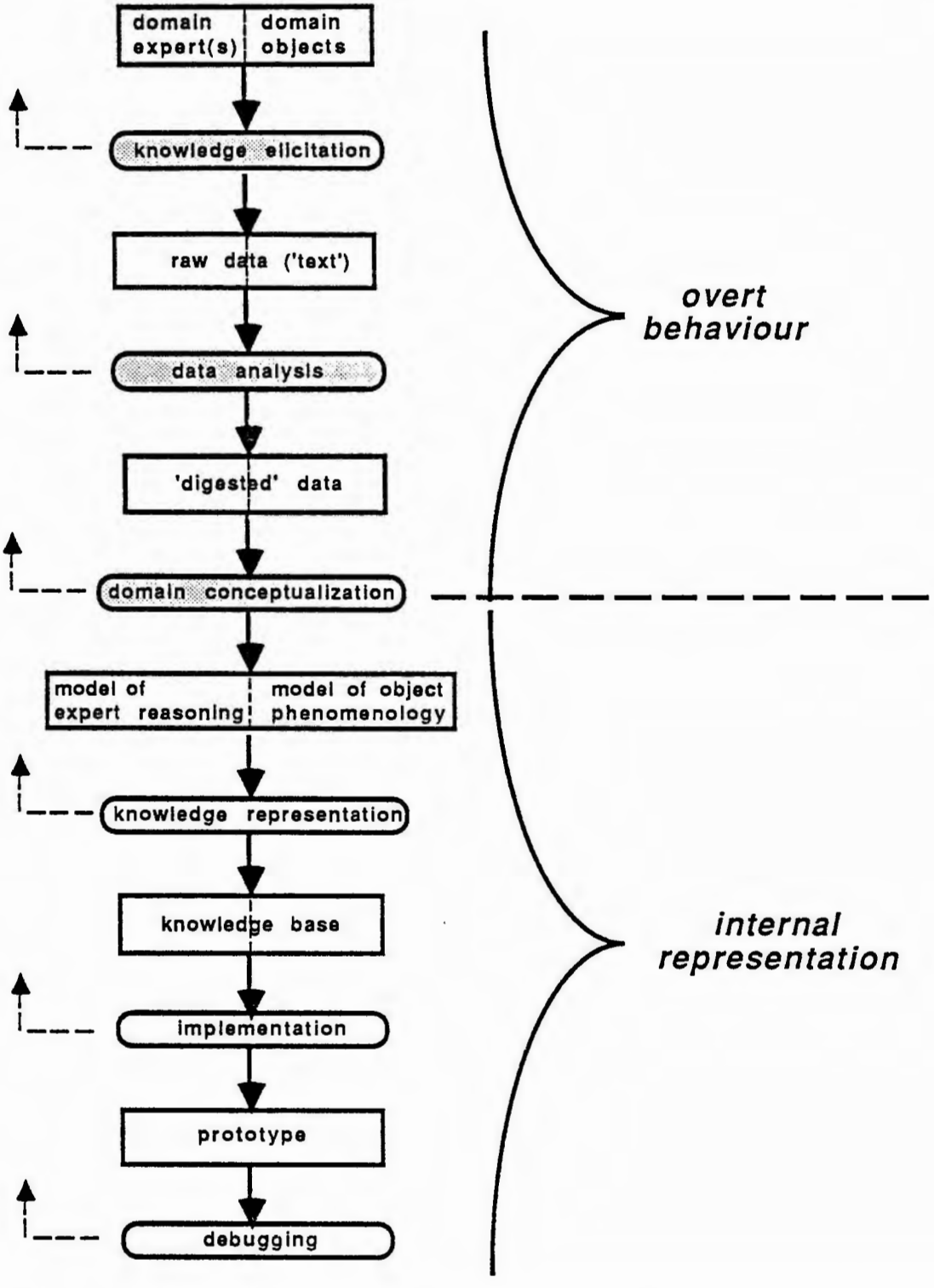


Figure 1. Functional view of the 'stages' involved in knowledge engineering. The shaded ovals define the knowledge acquisition phases.

2.2 Stages of knowledge acquisition

The preliminary phase of extracting knowledge from the expert usually produces an unstructured and uninterpreted set of data that are often messy, noisy and unreliable, depending on the particular technique being used. Typically, the more general the technique (such as interviews), the more unstructured and noisy are the elicited data, referred to here as 'raw data'. Figure 2 provides an example of the sort of raw text one usually gets from preliminary interviews with the expert. The figure illustrates part of a transcript obtained from interviewing an expert in a neurobiological domain.

> Mike: Because, I mean, the main thing that I'm interested in doing is to look at the, well, to do 2 things: one is to look at the cells which are changing when the animals are learning something new, to ask questions about how the patterns of dendritic connections are varying when the animal learns, to see how the spatial dimensions of the dendritic processes are changing, to see how their orientation, their field (which is this whole sort of area that they're growing in) to see how that changes, and to look at the connections that occur between the dendritic processes of one cell and the axons of another cell, or it may even be the dendrites of another cell...

Figure 2. Portion of a transcript of a knowledge elicitation session in the domain of neurobiology. This transcript is 'raw data'.

Figure 3 shows an excerpt from a journal article which serves as part of the knowledge engineer's 'homework'. We call this 'homework' because the knowledge engineer needs to gain some familiarity with the domain (exactly how much is an open research question), and therefore the study of background literature is an integral part of knowledge engineering. Other researchers, e.g. (Wielinga and Breuker, 1986), describe comparable 'homework' activities as forming an important part of the 'preliminary interview' phase.

> Formation of a memory for this one-trial passive avoidance task is unaffected by subconvulvie transcranial electoshock when applied 10 min. after training in 60% of birds, whereas "immediate" post-training electroshock renders 63% of chicks amnesic. Memory formation and retention is associated with a large bilateral enhancement in trained over control chicks (320 and 350% in left and right hemispheres, respectively; $p < 0.001$) of a particular spontaneous multi-unit firing pattern, that is, short-duration (15-40 ms) bursts of large-amplitude ($\geq$ 200 μV. 450 μV max p-p), high-frequency (400-450 Hz) spiking in anesthetized chicks.

Figure 3. Portion of relevant journal article (Mason and Rose, 1988) in the domain of neurobiology. This article also serves as 'raw data', and constitutes part of the knowledge engineer's 'homework'.

The most important aspect at the knowledge elicitation level is clearly the technique being used. In particular, there are trade-offs concerning the use of formal vs informal approaches, direct vs indirect methods (Olson and Reuter, 1987) and between weak (domain independent) and strong (domain dependent) techniques. Typically, knowledge engineers tend to use informal, direct methods, such as interviews. This style of elicitation has some advantages, as it suits industrial environments and it allows the expert to express a large amount of knowledge, as s/he is not constrained by a particular technique or representational encoding. On the other hand, the result of an unconstrained interview is usually a very unstructured set of data ('raw data'), that can require a major effort from the knowledge engineer to be analysed and conceptualized. Another problem, well analyzed in (Burton et al., 1987), is that the personality traits of the expert, for instance whether s/he is an introvert or an extrovert, also affect the elicitation of knowledge by interviews. Therefore, formal techniques, usually based on psychological principles, are also being used, in order to put knowledge elicitation on a firmer basis and to avoid getting unstructured and hence messy data. These techniques include: multidimensional scaling (Shepard et al., 1972), repertory grid (Boose, 1985) and hierarchical clustering (Johnson,

1967). Although the assumptions and the methods are different for the the various techniques, all of them attempt to produce a precise representation of the way the expert organizes his/her knowledge. For instance, multidimensional scaling produces n-dimensional diagrams that give an account of the similarities and the differences between concepts according to different relations. The main problem with these techniques is that they don't really allow a knowledge engineer to build an expert system. Techniques such as multidimensional scaling or repertory grid only elicit declarative knowledge. Analogously, think aloud protocols (Ericsson and Simon, 1984) primarily aim at eliciting procedural knowledge.

An important practical implementation of a formal knowledge elicitation technique is ETS (Boose, 1985), that uses the repertory grid technique. ETS asks the user to enter a list of domain *elements* first, and then to structure them, by differentiating triads of elements in terms of a trait that is true only for two of them. For instance, if the domain is a list of databases, a possible trait can be 'Run on Vax' or 'Hierarchical'. The combination of a trait and its opposite is referred to as a *construct*. Once a number of constructs have been elicited, the system asks the user to rate each element against each construct, therefore producing a rating grid. Finally, the system produces an entailment graph, that shows the implications between halves of constructs. These implications are then transformed into rules that, together with those generated from the rating grid, produce the final rulebase. ETS has the advantage of providing very fast prototyping (two hours) and of making the construction of the prototype completely automated. On the other hand, its technique seems suitable only for classificatory problem solving.

Finally, as the above mentioned techniques don't reflect the structure of a particular task, they are said to be *weak*. Researchers have found it useful to use strong models of knowledge elicitation, based on the structure of the domain and/or of the task. KNACK (Klinker et al., 1987) has knowledge about the design of electromechanical systems and uses such knowledge to make sense of the data it elicits from written reports and from interviews with the domain expert. A more interesting approach is found in MOLE (Eshelman and McDermott, 1986), where a heuristic classification model (Clancey, 1985) is used to elicit and interpret the data. MOLE implements the idea that the architecture of an expert system should rely on a more principled framework than a mere collection of frames and rules. For instance, heuristic classification models perform their reasoning in terms of general concepts such as 'data', 'solutions', 'abstraction' and 'heuristic step'. *Generic task models*, as advocated in (Chandrasekaran, 1987) are also based on a similar approach. Modelling problem solving in terms of generic tasks has a number of advantages. These models provide principled explanations (Clancey, 1983), based on the generic control structure of the problem, and they are more flexible and general. Generic task models can also be applied to knowledge acquisition (Wielinga and Breuker, 1986) to support top-down approaches. In section 4.3 we shall return to this point to discuss top-down support in Acquist.

Once a set of data has been elicited, the knowledge engineer attends to the data analysis activity with the aim of removing the noise and the unnecessary material from the raw data, by identifying and extracting atomic 'chunks' of knowledge in the form of concepts, statements, definitions or relations. The way data analysis is carried out heavily depends on the knowledge elicitation technique. Hence, it is relatively straightforward, when formal techniques are used, but it can be problematic if the raw data are particularly unstructured.

Data analysis is characterised by the emphasis placed upon the linguistic aspect of the data. For instance, preliminary data analysis of a report is carried out in KNACK by identifying and isolating keywords and concepts in the text, and in KRITON (Diederich et al., 1987), by means of a statistical text analysis. A similar approach is also expressed by Breuker and Wielinga (Breuker and Wielinga, 1985), where the authors discuss a linguistic level, characterised by the process of knowledge identification. At this level "The representation is superficial and has not been integrated into a conceptual framework. The same knowledge of different experts may have to be represented differently, because they use different terminology, or because their knowledge is structured differently" (Breuker and Wielinga, 1985). Hence, a typical outcome of the data analysis stage is the

production of glossaries and loose categorizations, such as the one showed in Figure 4, forming what we refer to as 'digested data'.

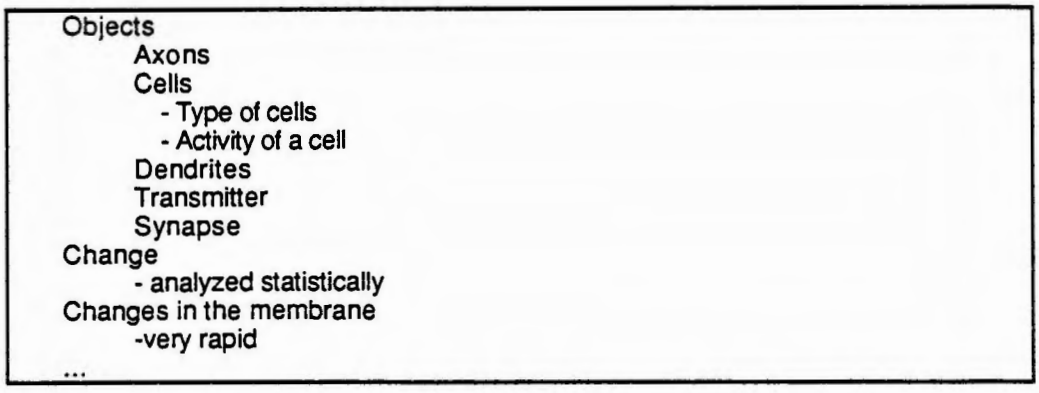


Figure 4. Atomic 'chunks' forming part of the so-called 'digested data'.

The categorizations produced at this stage do not need to be concerned only with the level of domain concepts. Figure 5 shows an attempt at a categorization of generic meta-level concepts, which can be used to account for the expert's behaviour in a neurobiology example. The purpose of this generic categorization is to pull out the concepts needed to build abstract models of scientific reasoning.

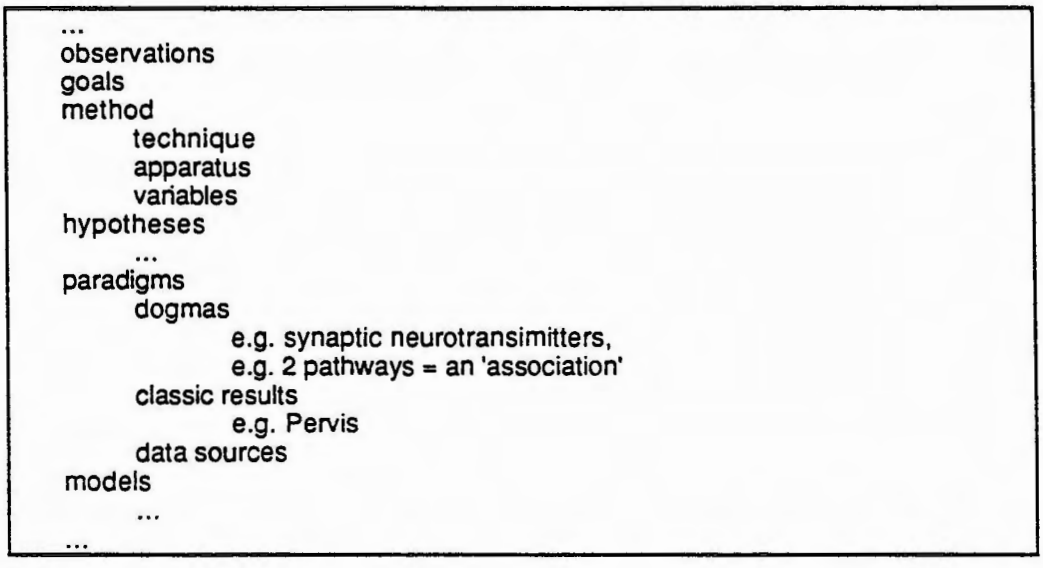


Figure 5 Generic concept groupings

It is worth stressing the fact that the so called 'digested data' produced by this analysis only exhibit a local meaning, as they are not bound together by a comprehensive model of the domain and the task. In fact, they express isolated chunks of knowledge that the expert can recognize, while not yet providing a model of the problem. This data analysis activity is useful for helping the knowledge engineer to home in on important aspects of the problem.

The production of glossaries and loose categorizations is only a part of the data analysis activity. More generally, as the aim of the linguistic analysis is to impose a structure upon a piece of a text, the outcome of this process should be the production of a linguistic model, characterized by a web of linguistic relations between textual entities. In sections 3 and 4, we shall describe what kind of support is provided in CREF and Acquist to produce linguistic models.

At the domain conceptualization stage, the knowledge engineer attempts to impose a structure upon the data collected so far in order to produce an abstract model of the problem in terms of taxonomic hierarchies, causal networks, tables, flow diagrams, or whatever organization s/he finds convenient for modelling the domain and the problem solving structure of the problem. The important element that differentiates this level from the knowledge representation one is that the representation at this level doesn't need to be runnable, but is meant to be a semi-formal characterization of the structure of the task. For example, Figure 6 shows a fine-grained characterization of 'event sequences', based upon the journal article excerpted in Figure 3. The representation is pseudo-code, i.e. it is not intended to be runnable as shown, but is intended to map directly onto a runnable representation.

```
event-class peck1, ; CHICK PECKS AT BRIGHT BEAD
        types: [bead(X), exp(E), chick(C)]
        salient-relations: [ ]
        sequence: [present(E,X,C)-1, peck(C,X)-2]
        replicability: good
        prior-events: [necessary: [] excluded: [] ] .

event-class avoid1, ;CHICK AVOIDS POISON BEAD
        types: [bead(X), exp(E), chick(C), aversive(Chem)]
        salient-relations: [coated(X,Chem)]
        sequence: [present(E,X,C)-1, avoid(C,X)-2]
        replicability: good
        prior-events: [necessary: [disgust1] excluded: [] ].

event-class shock10, ;10 MIN SHOCK HAS NO EFFECT
        types: [bead(X),electroshock(S), exp(E), chick(C)]
        salient-relations: [coated(X,Chem), delay-before-zap(10)]
        sequence: [prior-events-1/5, zap(E,S,C)-15,
                present(E,X,C)-16, avoid(C,X)-17]
        replicability: .6
        prior-events: [necessary: [peck1,disgust1,avoid1] excluded: [] ].
...
```

Figure 6. Domain conceptualization at a fine grain of detail, based upon an analysis of the journal article excerpted in Figure 3.

An example of conceptual structures is found in the interpretation models used in KADS (Breuker and Wielinga, 1985) and in ROGET (Bennet, 1983), that represent the strategic layer of the task. Indeed, there is no reason for conceptual models to be characterized only at the meta-level. In fact, in domains that exhibit complex device behaviour, such as electronic fault diagnosis, most of the action actually lies at the domain level, i.e. in understanding the functionality and phenomenology of the devices, while the strategic layer is often much less complex. We have underlined this duality in Figure 1 by splitting the outcome of the domain conceptualization stage into two ports, dealing with the strategic level ('model of expert reasoning') and the phenomenology of the objects in the domain.

Domain conceptualization is a very complex and delicate stage of the knowledge engineering process, as it is at this level that inconsistencies and holes appear in the knowledge base, although, given the semi-formal nature of the models, they can't automatically be spotted. Another problem concerning top-down approaches is that when conceptual models are used to drive the acquisition of knowledge, it may be the case that the model and the domain don't match. This is especially true for systems that implement automated knowledge acquisition (see for instance MOLE (Eshelman and McDermott, 1986)), as they can only map the domain dependent data into the prestored model and provide little help when such a mapping is not feasible. To make things more complex, the expert may not be fully aware of his/her assumptions and problem solving strategies. Therefore, it may be up to the knowledge engineer to uncover and make explicit the rationale supporting the elicited data. For instance, in (Motta et al., 1988) we showed some examples of common-sense based strategies used in

fault diagnosis. These strategies were inferred inductively by the knowledge engineer after an analysis of the rules describing the behaviour and the diagnostic choices of the expert.

2.3 Knowledge acquisition as the production of conceptual models

The 'stage' model in Figure 1 leads to a view of knowledge acquisition as the production of an abstract architecture that is distinct from the implementation of the system. This view has some advantages that include the factorization of the tasks and the possibility of examining the strengths and weaknesses of the model before implementing it (Wielinga and Breuker, 1986). On the other hand, this characterization is also problematic, as it seems to clash with the problem of devising methods for checking whether knowledge has been acquired and whether the abstract architecture is a real model of the problem. In fact, the only way we know of testing an expert systems application is to run it. Only then is it possible to check whether the appropriate knowledge has been operationally acquired. Therefore, even if it is methodologically useful to characterize knowledge acquisition as an activity whose aim is to produce a conceptual model, we believe that in practice the boundaries between acquisition and implementation can actually be very fuzzy. Clearly, this is due to the lack of an appropriate formalism and of primitives for conceptual structures. Whether such primitives can be found, is an open problem.

Another issue concerns the nature of the model specification. As mentioned before, we agree with the view expressed in (Wielinga and Breuker, 1986) where the authors advocate the use of models at the 'epistemological' level, based on the 'generic tasks' paradigm, both to drive and to characterize the outcome of the knowledge acquisition process.

2.4 Supporting the knowledge engineer

Most commercially available AI toolkits only support the knowledge engineer during the phases of implementing and debugging the system. No support is normally provided at the knowledge acquisition stage. On the other hand, a number of efforts have been made toward the direction of automating the knowledge acquisition process (Davis and Lenat, 1982; Boose, 1985; Bennet, 1983; Eshelman and McDermott, 1986). For better or worse, automated knowledge acquisition hasn't yet reached the point of making the knowledge engineer redundant, and, for most applications, it is still necessary for the knowledge engineer to tackle all of the activities described in our model. To understand why this is the case, consider an analogy between knowledge engineers and traditional programmers. In the case of programmers, although life can certainly be made more pleasant by the provision of syntax-directed editors, smart compilers, and good debugging environments, it seems unlikely that an automatic programming system can help in the design and conceptualization phase of anything other than tasks that have been previously 'solved'. Analogously, in the case of knowledge engineers, life can be made more pleasant by the provision of good life-cycle tools, but it seems to us unlikely that anything approaching automated knowledge acquisition/representation will be forthcoming for domains that have not already been 'solved'.

These considerations have provided the basis for our work on the KEATS project. KEATS as a whole aims to provide life-cycle tools for the parts of Figure 1 shown in ovals. In the next section we provide a brief description of the support for knowledge acquisition provided by the first KEATS prototype, KEATS-1. Following that, we introduce the new life-cycle tools being developed as part of KEATS-2.

3. BRIDGING THE GAP BETWEEN KNOWLEDGE ELICITATION AND IMPLEMENTATION: KEATS-1

The development of KEATS (The Knowledge Engineer's Assistant) was motivated by the idea of building a knowledge engineering toolkit that could provide a comprehensive range of tools to help the

knowledge engineer fill the gap between the raw data and the final system. The first implementation, i.e. KEATS-1, incorporates not only a variety of typical representational aids, such as frame and rule languages, but also non-standard facilities such as a package for transcript analysis and an 'intelligent' graphical interface, to (partially) support the analysis and structuring of the domain. In particular, as far as knowledge acquisition is concerned, KEATS-1 provides a Cross Reference Editing Facility - CREF (Pitman, 1985), and a Graphical Interface System - GIS, to support the data analysis and the domain conceptualization stages of the process. These are described in turn below. The interested reader can find a more detailed discussion of these and other KEATS-1 facilities in (Motta et al., 1988).

3.1 CREF: An editor for transcript analysis

A knowledge engineer often has to perform transcript analysis. Many knowledge elicitation techniques, including popular ones such as interviews and protocol analysis, require the knowledge engineer to analyze text in order to extract the elicited knowledge. Text analysis is also performed when gathering knowledge from other written material such as journal articles, books or documents. This type of analysis is usually a time-consuming activity for which AI toolkits normally provide very little support. Therefore, it is worth asking what sort of tools are needed to help the knowledge engineer speed up this phase.

Typical editors support text processing within a unidimensional presentation mode that can be related to conventional documents or to a particular programming language; in fact, editors support *synthesizing* activities, such as writing programs or creating documents. In knowledge engineering, however, the activity is mainly *analytical*, as the knowledge engineer processes text while aiming to understand an unstructured collection of data. Therefore, it has been recognized (Anjewierden, 1987; Diederich et al., 1987) that knowledge engineering tools require dedicated, non-conventional text editors that can support the process of organizing, abstracting and interpreting data from a text.

The observation of knowledge engineers at work shows that when they organize textual material, they attend to mundane activities such as recognizing and highlighting important words, lines and paragraphs, producing references, annotating and grouping together related chunks of text. CREF allows the knowledge engineer to organize the text by partitioning it into atomic entities called 'segments' and grouping them into 'collections', according to a number of rationales that can be either syntactic (string matching) or semantic (summaries, logical connections, model dependent keywords). The result of this editing oriented activity is then a set of collections that defines a model of the problem at the linguistic level. Such a model is usually rather sketchy and empirically motivated and constitutes the preliminary stage of the data analysis activity. Nevertheless, it is a good basis for facilitating further data analysis and for providing a framework in which to conduct additional knowledge elicitation sessions.

3.2 GIS: A Graphical Interface System

In order to help the knowledge engineer produce a conceptual model from the linguistic level structure provided by CREF, KEATS-1 provides him/her with GIS. GIS makes it possible to build a graphical representation of the domain by drawing diagrams onto a 'sketchpad'. These diagrams are made up of graphical objects that may be used to represent any type of entity, from CREF segments to objects in the domain under examination. As the diagrams are automatically translated into a frame language, they are made operationally meaningful and facilities such as inconsistency handling are supported. Thus, GIS' behaviour is akin to that of an active blackboard, which is able to make sense of the diagrams being drawn. In fact, while providing a high level medium that frees the user from the idiosyncrasies of a particular knowledge representation language, it also gives him/her interactive support: the consistency of the evolving model is checked, frames are created and constraints propagated.

GIS provides an input-driven user interface, as the user directly creates and manipulates graphical objects. This is different from other graphical interfaces that permit the user to have a network or tree-like view of the knowledge base automatically drawn by the system. While such tools have their utility, they suffer from being based on *a posteriori* displays of the knowledge base. In contrast, GIS allows the user to work directly on the interface, to control decisions concerning the layout and to selectively view the objects and the relations that are relevant to the problem at hand.

3.3 The limits of CREF and GIS as tools for knowledge acquisition

The aim of CREF is to support the knowledge engineer in the analysis of transcripts, by providing an editing facility that can help him/her impose a structure upon arbitrary pieces of text. CREF supports this task with facilities that can be thought of metaphorically as a means of weaving arbitrary coloured threads through pieces of text (segments) and then allowing the user to pull on the threads at whim in order to obtain selective views (collections) of the transcript. However, in terms of the CREF display, the views produced by pulling on the coloured threads are just another linear representation of the knowledge contained in the transcript and as such limit the structuring and abstraction of that knowledge. So, although CREF can be seen as a step forward from typical editors such as EMACS (Stallman, 1979), much of its user interface is still 'traditionally' teletype oriented and doesn't fully support the aforementioned threads metaphor.

Another problem experienced in using KEATS-1 is the lack of an integrated support for knowledge acquisition. Data analysis and domain conceptualization are distinct but strongly interrelated tasks, and the output of the first activity is normally the input to the second. Therefore, it is natural for a knowledge engineer to expect knowledge acquisition tools to support this relation of continuity. In particular, one requires from a domain conceptualization tool the facility of accessing, modifying and reasoning about the chunks of knowledge produced by a data analysis tool. In the combined CREF/GIS system, this is possible only indirectly, as the two tools work with different sets of objects: segments and collections in the CREF case, graphical icons and diagrams in the GIS case. The aforementioned 'continuity' requirement is fundamental for supporting the smooth progression and the incremental refinement of models implied by Figure 1.

For instance, a typical operation that a knowledge engineer performs upon a piece of text is that of isolating a rule, fact or concept. A knowledge acquisition support tool should then allow the specification of links from the newly created objects to the original piece of text. This allows such a structure to be used for generating automatic documentation/explanation, which records the rationale underlying the definition of such an object, or alternatively to be used for the sake of bookkeeping. A simple operation such as the one described here is possible in CREF/GIS only if the knowledge engineer specifies this link indirectly, which requires a number of steps to duplicate structures across the two systems.

The problems experienced in the use of CREF and GIS have caused the KEATS-1 knowledge acquisition module to be re-evaluated and have led to the design and development of Acquist, a knowledge acquisition tool to be embedded in 'KEATS-2'. KEATS-2 is itself an evolving software research environment, intended to take the lessons learned from KEATS-1 and provide support tools suitable for scaling up to the size of problems faced by teams of (possibly disparate) knowledge engineers. Acquist is expected to assist the knowledge engineer in both 'bottom-up' and 'top-down' activities. In the former the knowledge engineer 'feels his/her way' around a domain for the first time, and in the latter the knowledge engineer tries to verify or instantiate or 'flesh out' the skeleton of pre-existing schemas by observing the expert in action or discussing the domain in detail with the expert.

We encourage a mixture of styles, depending on the problem at hand and the user's own preferences. Purely top-down regimes can be uncomfortable for some users, in that they appear to 'steamroll' the user into a pre-determined style of representation. Purely bottom-up regimes can be too *adhoc* for some users. KEATS-2 aims to allow the user to build up an account of expert behaviour in a domain

in a bottom-up fashion, then to apply that account top-down to further examples of (the same or related) behaviour, or to freely intermix the two styles. The next section describes out latest efforts in this area.

4. ACQUIST

4.1 Introduction

Acquist is a hypertext based facility (Conklin, 1987) that allows the knowledge engineer to carry out knowledge acquisition through the process of abstracting and structuring the knowledge contained in 'raw text' (typically transcripts). Units of knowledge are isolated from the raw text through a process of defining chunks of text called 'fragments' (data analysis). Structuring of the knowledge contained in fragments occurs at three different levels. Firstly, fragments are collected around 'concepts'. A concept can be thought of as the abstract label for the notion that the fragments in that collection define, or concretize. Secondly, concepts can be factored into 'groups', and 'groups' into 'meta-groups', which split the concepts into higher order categories. The third and most flexible means of structuring the knowledge is a means of defining links and applying these links between 'items' (i.e. fragments, concepts or groups) either across the same level of abstraction, or, as is the case with the 'documentation' link, across different levels (domain conceptualization). A collection of fragments, concepts, groups and links constitutes a model or theory of the knowledge for a given domain, and can be saved by the user as a 'theory'.

In addition to its focus on text, Acquist also provides a generic graphical layout facility which we refer to as 'maps'. Maps provide a graphical representation of the emerging structure which is being imposed on the text, and equips the knowledge engineer both with a high level view of the knowledge extracted, and a means of navigating through the transcript. Maps also provide a graphical interface to the acquisition process, allowing the knowledge engineer to carry out all but the fragment generation operation via a graphics rather than text based interface. More importantly, the graphical layout produces an intermediate level of representation, bridging the gap between knowledge acquisition and the implementation of the system.

While the CREF/GIS modules in KEATS-1 were only able to (partially) support bottom-up approaches to knowledge acquisition, Acquist aims also to provide top-down support, by means of a mechanism for on-line creation and completion of 'coding sheets'. These are templates for driving the knowledge acquisition process, and will be discussed in section 4.3.

Acquist aims to integrate both the philosophy and the functionality of the two original KEATS-1 facilities, CREF and GIS, in order to produce a more natural and more efficient progression from the transcript, through data analysis, to domain conceptualization, and finally, to the formalization of the knowledge into a machine readable form. Acquist builds upon the lessons learned in the implementation and use of CREF, and like CREF, its design aims to support the tasks that knowledge engineers have been observed carrying out when analysing transcripts. These tasks include, editing/revising the transcript to filter out uninteresting and irrelevant parts of the transcript; segmentation of the transcript into logical units of information (fragments); indexing of fragments; highlighting of particularly important information; annotation of the fragments; construction of diagrams; generation of models from the information held in the transcript.

However, Acquist differs from CREF in that the emphasis is placed upon creating conceptual models based upon the fragments of text that the knowledge engineer deems to be important. Thus, the processes of data analysis and domain conceptualization are not kept separate but rather become intertwined, allowing the knowledge engineer to work at both levels at will, by chunking the transcript into meaningful pieces of knowledge, asserting links between them, and at the same time generating concepts and relations in the domain. This activity leads to the construction of an abstract architecture of the domain that is *tied to* and *supported by* the fragments of text from the transcript. Moreover,

from the abstract architecture, the user can automatically generate a frame-based representation, which can then be used as the kernel of the prototype system. This is precisely the 'bridge' between overt behaviour and internal representation emphasized in our discussion of Figure 1 earlier.

We now turn to a more detailed discussion of the structure and facilities provided by Acquist.

4.2 Providing support for bottom-up approaches

4.2.1 Transcripts

Acquist deals first and foremost with raw text obtained either from interviews with domain experts or from relevant domain literature. The transcript is read into Acquist using the command 'Read Transcript' (Figure 7 - top right) which provides the knowledge engineer with access to a read-only version of the text. This ensures that any future analysis of the transcript, by the same or other knowledge engineers, is based on exactly the same text.

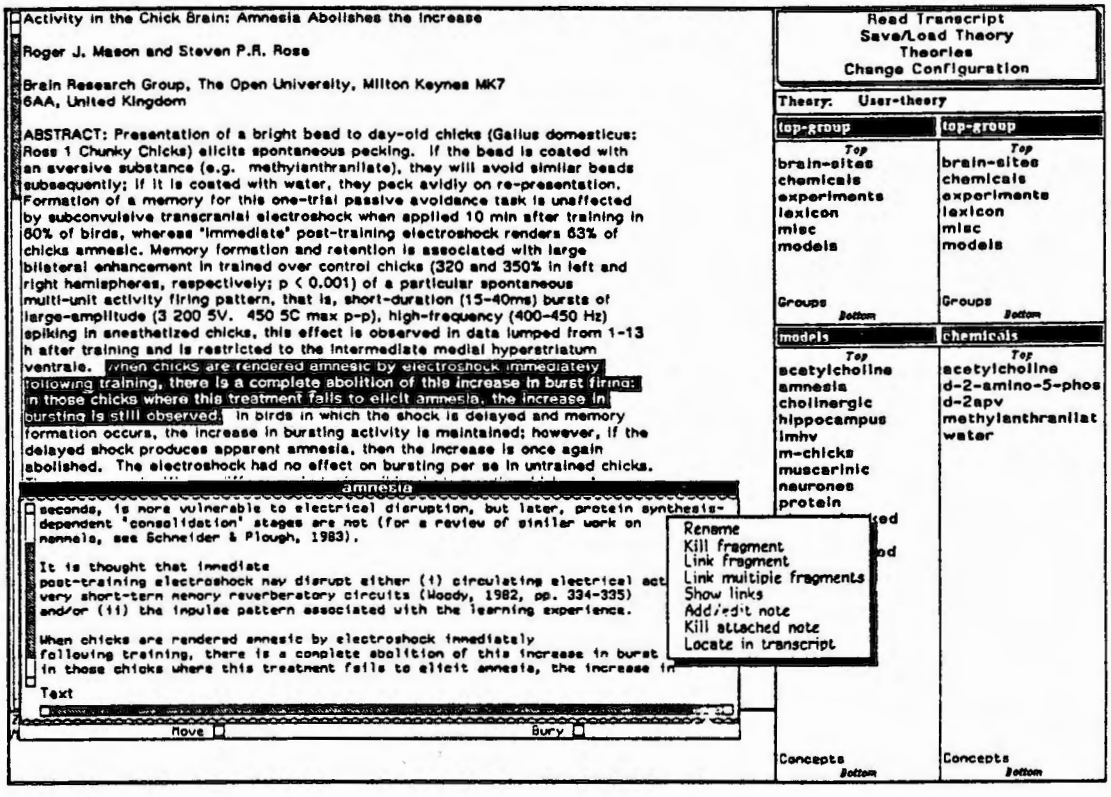


Figure 7. Screen snapshot of Acquist. The top right hand corner of the display contains a command menu, which provides the knowledge engineer with the means of reading in a new transcript, loading and saving theories (the work done during acquisition) and changing the Acquist configuration. Notice the pair of 'concept' and 'group' menus on the right hand side. These are discussed in sections 4.2.3 and 4.2.5

4.2.2 Fragments

The basic textual unit manipulated by Acquist is called a 'fragment', which is an arbitrarily chosen chunk of text taken from the raw text base. A fragment can be any size from a letter or a sentence to a whole transcript. It is left to the knowledge engineer to decide which parts of the transcript are important, and what the boundaries of this information should be. It is possible to create fragments

that overlap, i.e. have common pieces of text, and also for fragments wholly to contain text used in other fragments.

Fragments are created when the user selects a region of text from the transcript, and assigns it to a concept by mousing on the relevant concept name in one of the two concept scroll menus. Figure 7 shows a screen snapshot of Acquist in action. On the far left, a transcript which is being analysed is displayed in a specialized text editor. A fragment chosen by the knowledge engineer can be seen highlighted in inverse video. Also shown in Figure 7 is a pop-up window displaying the contents of the fragments clustered under the concept 'amnesia'. This was achieved via a pop-up menu which resulted from the user selecting the concept 'amnesia' from the scrolling concepts menu. A mouse-click on any of the fragments allows the user to apply a number of fragment-specific operations, such as scrolling the transcript directly to the chosen fragment.

Fragments are created by selecting a chunk of text with the mouse and then clicking on the concept which categorizes that fragment. That fragment then becomes part of a collection of fragments which together embody the meaning of that concept. In Figure 7, a list of defined concepts reside in two scrolling menus at the bottom right of the display.

4.2.3 Concepts

A 'concept' can be thought of as a label representing a class of items, in this case fragments of knowledge, that contribute to a concrete description of that concept. Concepts can be used in two ways, top-down and bottom-up. In the former case (top-down) the knowledge engineer starts off with a particular concept and searches through the transcript to find fragments which are relevant to the concept. In the latter case (bottom-up) the knowledge engineer finds an interesting fragment and thinks up an appropriate concept which classifies it. Concepts can be generated by hand or from an automated lexical analysis of the transcript, as described in section 4.2.4.

When concepts are created, it is possible, at the same time, to create a frame to provide an internal representation of that concept, so that the user can work at different levels at the same time. When there is a 1:1 mapping between concepts, icons and frames, an operation at each of the editing, graphical or implementational levels affects the environment at the two other levels. This is especially important in order to give the user a correct view of the knowledge base, as changes at the frame level automatically update the graphical interface.

4.2.4 Lexical analysis of the transcript

As an easy entry point into knowledge acquisition in a new domain, and as a means of identifying important domain concepts, Acquist provides a first pass lexical analysis of the transcript(s). The knowledge engineer chooses a transcript or set of transcripts and can use them to create a new lexicon, or add to the existing one. This lexical analysis produces a list of all the different words that are used in the transcript. However, because many of the words will be uninteresting and irrelevant, the knowledge engineer can also create a filter to extract these uninteresting words from the lexicon. The filter is created or added to by reading in an ordinary text file of the words that the knowledge engineer is not interested in. Sets of filter files will be available for knowledge engineers to produce a concise lexicon. Such a set might include, adjectives, determiners, names, numbers, verbs, and possibly domain specific words.

When the lexical analysis has been carried out, all the words generated are placed into the group 'lexicon', which then acts as a pool of concepts. The knowledge engineer can then peruse this list and factor into 'groups' those concepts that might be of interest. S/he can also kill those uninteresting words that were not extracted by the filter. Concepts can then be used as keywords with which to access the transcript, i.e. the knowledge engineer can visit each place in the transcript where that concept or word occurs, possibly creating fragments as s/he proceeds.

4.2.5 Groups

A 'group' can be thought of as a higher order concept that provides an abstract rationale for a number of concepts. For example where a concept is a class, a group is a meta-class labeling that class of concepts. In practice groups are used in the same way as concepts, so there is no reason why groups cannot contain other groups as well as concepts. The main use of groups is in factoring concepts into meaningful categories, making the acquisition of knowledge from large domains easier to handle. This process can be taken as far as the knowledge engineer wishes, as groups can also be factored into higher order groups. Two group menus can be seen directly above the concept menus in Figure 7. The name of the group whose contents is on display can be seen at the top of the corresponding group menu.

When the knowledge engineer selects a group from either group menu with the mouse, the concepts in that group get displayed in the concept menu directly beneath, and the name of the selected group becomes the title of that concept menu. In order to change the group whose contents is being displayed in both the group and concept menus, the knowledge engineer must click the mouse on the group title. S/he is then presented with a menu of either that group's parents, or all its superior groups. The groups and concepts of the selected group are then displayed in the appropriate menus. The reason for having two concept and group menus is to facilitate the linking and movement of concepts. Since Acquist depends heavily on the use of easily-expressed binary relations, the provision of two concept and group menus is more expedient than just one.

There are three system defined groups, 'top-group', 'lexicon' and 'misc'. The lexicon is used to store the words generated by the automated lexical analysis, while the misc group is provided as a scratch pad to place concepts for which the knowledge engineer has not yet found a home. The top-group is the highest level group, ie the most superior group.

4.2.6 Relational Links

Acquist allows the user to link items (i.e. fragments, concepts, groups) in several different ways. These links allow the creation of linguistic models based on relations between textual objects and to link textual objects to other non-textual objects. Besides, the user can impose a particular structural organization by defining new types of links. These facilities provide the Acquist user with a flexible means of abstracting and structuring the knowledge contained in the transcript(s). The following set of domain independent links, based on our own experience in analyzing documents, is provided for linking items together:

- NOTES can be attached to items so that the knowledge engineer can make some comment about the item in question. The note may be a summary, a reminder to ask the expert a question, or extra information about the topic contained in the item. The way in which notes are used is left up to the user.

- SPECIAL-CASE links relate one fragment to another that contains a more general restatement of a concept, a class of a relation.

- JUSTIFIES links are used to link some statement in the text to a fragment that provides a justification for it.

- EXPLAINS links relate a piece of text to a fragment, that clarifies or gives the rationale for it.

- REINFORCES links relate an assertion to a fragment that contains some information strengthening the assertion.

- WEAKENS links relate an assertion to a fragment that contains information weakening the assertion.

- INCONSISTENT-WITH relates two mutually inconsistent fragments.

- DOCUMENTATION links allow fragments to be used to document a concept. If this has a frame attached, the fragment can be used as the documentation for the frame as well.

The user can also define links to allow a wider range of relationships to be made between items. The knowledge engineer is thus not restricted in the way s/he is able to represent the knowledge contained in the transcript. User defined links consist of a forward and a backward component. The forward component is the name of the link from item 'A' to item 'B', whilst the backward component is the name of the link from 'B' to 'A'. If either of the link components is left out then the link becomes unidirectional, while if both components are the same it becomes symmetric. In maps (see below) these links are represented by arrows between the graphical representations of the items. As with 'notes' there is a scroll window which shows whether an item has links attached to it. These 'notes' and 'links' menus are tied to their respective group and concept menus and move with them when they are scrolled. A mouse click provides a more detailed breakdown of the details of which links are used and which items are linked by them. A similar menu can be called up for fragments.

4.2.7 Maps

A 'map' is a graphical representation showing the topology of a set of concepts, groups or fragments. The relationships or links between items are shown by arrows between the graphical objects. These links can be specified by hand in Acquist or by means of the graphical interface. Maps do away with the intermediate linear view of collections used in CREF and provide the knowledge engineer with an immediate high level view of the structure that s/he is building. If the knowledge engineer wants to view the collection of fragments that define a concept whilst viewing a map, s/he just mouses on the concept, chooses the relevant menu option, and a pop-up window displays the fragments. Figure 8 displays an example of a map showing the relationship between a set of concepts.

The knowledge engineer can manipulate items through mouse and menu driven operations from either the graphical interface or the Acquist display. Maps are a very important part of the process of analysing transcripts, because they are a vehicle for producing conceptual models. One problem with transcript analysis is the huge amount of information generated. Maps provide a means of organizing that information. Once a map has been constructed, or is in a state of construction, it provides the knowledge engineer with a simple means of navigating through the mass of text contained in one or more transcripts. Moreover, the knowledge engineer is in charge of the decisions concerning the layout, and a number of facilities are provided for working on the displayed representations. These facilities are similar to those provided by GIS in KEATS-1 and include means for working with multiple but consistent diagrams, defining graphical links, moving groups of icons and links, duplicating graphical structures, and selecting long-distance and close-up views.

As mentioned above, maps provide the means for a smooth progression from the generation of an abstract description of the knowledge contained in transcript(s) to the generation of a frame based representation and ultimately an implemented knowledge based system. In this case, the 'overt behaviour'/'internal representation' gap (Figure 1) is effectively bridged. It remains for the knowledge engineer to specify an interpreter which can 'walk over' a given representation and behave correctly. In the most problematic (and interesting) cases, maps allow the specification of 'coarse-grained' descriptions, that can then be iteratively refined to produce more fine-grained ones. For instance, in (Motta et al., 1988) we show how different levels of descriptions can be integrated to account for the phenomenology of the domain both at a coarse-grained level (black-box devices) and at a fine-grained one (component circuit diagram).

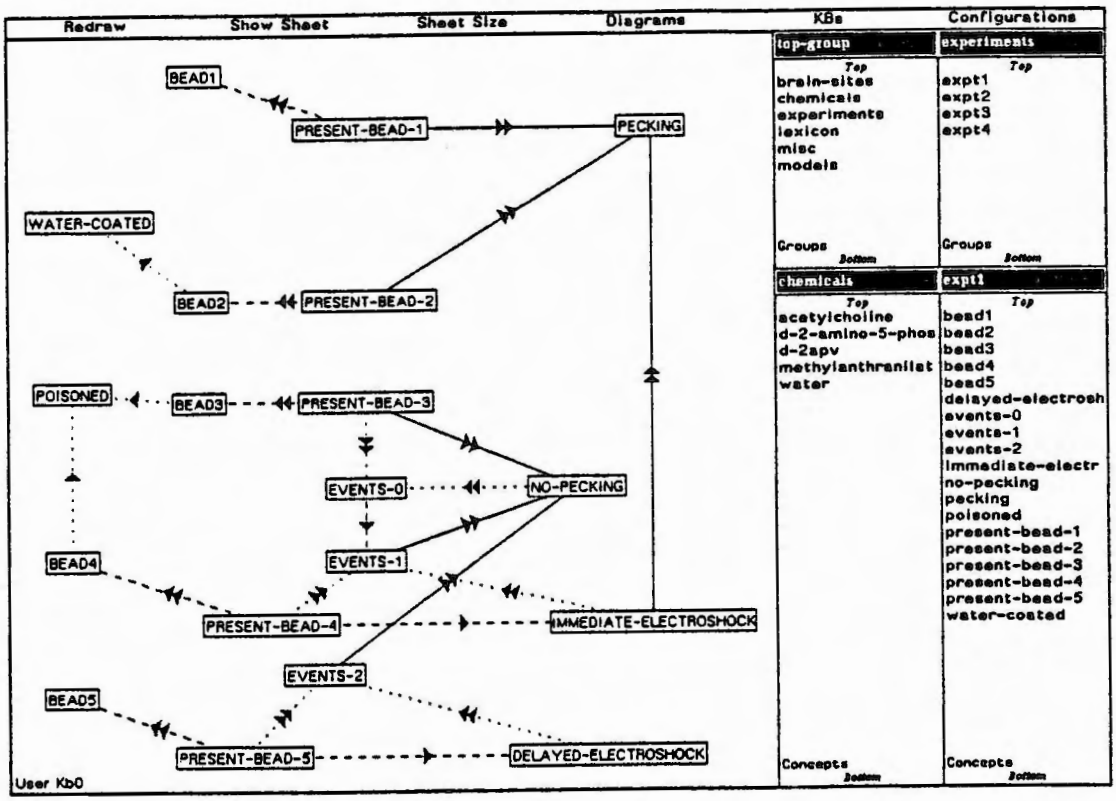


Figure 8. An Acquist 'map', showing the interrelation of selected concepts.

4.2.8 Theories.

Theories are the means that Acquist provides for saving collections of fragments, concepts, groups and links, which constitute some form of text based model of the domain. This is carried out with the 'Save/Load Theories' menu command. Acquist allows multiple theories to be loaded at the same time, so that the knowledge engineer can switch between theories at will. This facility also allows different knowledge engineers to work on the same set of transcripts, building independent models of the domain. Acquist also provides a means of merging 'theories' should this be necessary. The name of the current theory is shown just under the main command menu, in this case 'mason'.

A parallel feature, called knowledge bases, is provided by the frame based system embedded in KEATS-2. A combined text and frame based model can be saved and loaded as a 'model'.

4.3. Providing support for top-down approaches: 'Coding sheets'

So far we have shown how Acquist can lead to the development of a knowledge base, through the analysis of text and the development of a conceptual description of the domain. We have observed that in some cases the knowledge engineer may already have a predefined abstract model of the task and/or the domain that can be used to drive the knowledge acquisition process. In this section we introduce and describe the notion of 'coding sheet' and we show how it can be used to drive knowledge acquisition in a top-down fashion.

A coding sheet is a form on which to record a caricature of the observed behaviour of the domain expert. Knowledge acquisition systems base their behaviour upon weak or strong models of the problem, which are used to drive the knowledge elicitation session and to understand the data elicited. As the mapping between the problem and the preexisting model can be fairly loose, the acquired

knowledge base (and the eventual system) automatically built may not be a faithful representation of the problem. However, as experience shows, the use of preexisting models (even inadequate ones) can lead to dramatic improvements in the time required for performing knowledge acquisition and building a prototypical system.

We have found in practice that a carefully constructed coding sheet for a task such as electronic fault diagnosis has enabled a pair of knowledge engineers to conduct very rapid 'knowledge acquisition raids' on a domain expert. In our case, one knowledge engineer conducted the bulk of the interview while the other filled out the coding sheet in real time throughout the length of the six-hour session. The coding sheet form itself reflects the coarse-grained activities which occur during straightforward electronic fault diagnosis. Early versions of the form simply used paper and pencil variations, but we are now experimenting with 'rapid cross-indexing' versions, using the hypertext facilities of Acquist, as well as those of independent software tools such as HyperCard (Goodman, 1987). Any given coding sheet comprises a set of individual 'cards'. Figure 9 shows a single card from a coding sheet used for a fault diagnosis interview session.

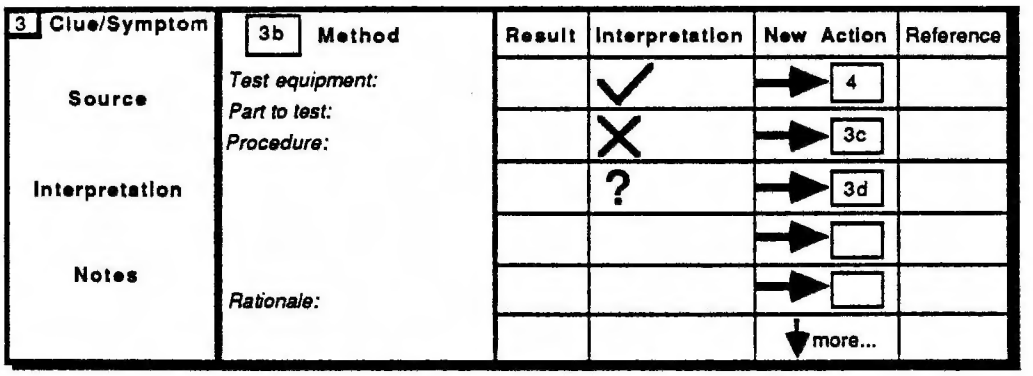


Figure 9. Coding sheet used for real-time encoding of expert behaviour on a fault diagnosis task. Sheet can either be paper-and-pencil variety, or true hypertext style with mouse-sensitive arrows for branching to new sheets.

The left-hand column of Figure 9 is a place for noting the current symptom and where it comes from (e.g. 'fault docket'). We use the word 'clue' to indicate that diagnosis is not just symptom-based, but may be based on the outcome of previous intermediate results. Indeed, in our hypertext implementation, the 'source' entry includes an automatic (mouse-sensitive) link back to the coding sheet result which led us here in the first place. The second column can be used to indicate the particular diagnostic technique. In the case of a hypertext implementation, such techniques can be carried on (pasted) from one coding sheet to the next to avoid tedious reduplication. The last four columns provide a means for recording any of six possible outcomes of a test, with the possibility of 'extending' the sheet by pointing off to 'more' results. Three standard interpretations are catered for ('win', 'lose', and 'don't know'), although these can be specified in more detail by the knowledge engineer. The new action is typically just a pointer off to more coding sheets, wherein the diagnostician follows through the initial interpretations.

In Acquist, each coding sheet is represented as a group, and each card is represented as a concept belonging to that group. The individual cell entries can be concepts, fragments, or links thereto. When the coding sheets are used in real time for knowledge elicitation, Acquist takes care of generating (a) new cards with links to previous ones, (b) the appropriate internal structures (i.e. concepts and fragments) for the cell entries keyed in by the knowledge engineer, and (c) links corresponding to specific coding sheet labels (e.g. 'Notes' and 'Source').

Another important use of the coding sheet is for retrospective detailed analysis of the transcript. Consider, for example, the case when the domain expert explains the interpretation of the 'lose' outcome of a particular test in great detail. The knowledge engineer can simply highlight the text of the entire explanation in the transcript, and mouse-click on the relevant cell in the 'Reference' column on the right hand side of the coding sheet. The result of this action is to leave a fragment symbol ('f') in the Reference column, which itself may be selected for performing fragment-specific operations.

The coding sheet shown in Figure 9 is at first glance nothing more than a way of encoding very large decision trees or discrimination nets without having to fill up enormous wall-charts. This in itself is not such a bad thing, as it concisely encapsulates a wide range of diagnostic behaviour. On second glance, however, it is much more flexible than an ordinary decision tree, because (like the production rule architectures which inspired it) there may be numerous entry points/paths into a particular node, which can be cumbersome to represent in the form of a tree. The main point we wish to make here is that a concise encoding of the knowledge elicited during an interview can facilitate later analysis, because it may be possible to encode at least a coarse-grained variant of the entire session in real time.

Coding sheets are mainly intended as an aid to yielding an analyzable set of 'digested data', as in Figure 1, although they may be useful in their own right if the entries are precise enough to be converted directly into an internal representation. Our own experience with the use of the coding sheet depicted in Figure 9 has shown that it proved effective in encoding most of the diagnostic behaviour of the expert and led to the rapid implementation of a prototype. However, the type of system so generated, organized around a decision tree diagnostic structure, was a 'shallow' one and, as such, of limited power. A much more powerful and principled implementation, as described in (Motta et al., 1988) could only be devised after developing functional and formal models of the domain and after an in-depth analysis of the expert's cognitive behaviour. Nevertheless, this experience convinced us of the advantages, in terms of the quality of the elicited data, provided by the use of strong (domain-dependent) techniques for knowledge elicitation.

4.4 Acquist, KADS and KRITON

Two other knowledge acquisition systems place the same importance on analysing transcripts in order to extract expert knowledge: KADS and KRITON. KADS supports both a bottom-up and top-down approach to knowledge acquisition, while KRITON only supports a bottom-up style.

KADS (Anjewierden, 1987) supports data analysis and domain conceptualization with two separate tools called PED and CE. PED is a hypertext based protocol editor, which provides facilities for fragmenting a transcript, creating links between fragments, grouping fragments together, attaching notes, and assigning fragments to concepts. The Concept Editor allows the knowledge engineer to create concept hierarchies with user defined links. Concepts can be given a list of attributes which can be inherited down through 'isa' links. Both the concept hierarchy and the fragment network can be viewed graphically to show the topology of the structure.

Top-down knowledge acquistion is supported through the provision of a library of interpretation models. An interpretation model consists of a set of inference structures describing the meta-level structure of a generic task. The knowledge engineer then chooses which interpretation model best fits the current domain, and applies it to that domain. Although the precise way in which these libraries are used by the user is unclear, the basic idea is that the product of the knowledge acquisition process is an interpretation model, that can be either constructed or selected from the preexisting library.

Hence, there are a number of similarities between KADS and Acquist, that include the emphasis on supporting the user at different levels, provisions for both top-down and bottom-up support, and the use of hypertext metaphors and graphical interfaces. A relevant difference is that Acquist provides a richer set of predefined links at the linguistic level, while KADS only predefines 'groups' and 'annotations' links.

From a more general point of view, the main methodological difference is that while KADS emphasizes the production of a 'documentation handbook' and the development of semi-runnable conceptual models, Acquist is itself part of a software environment (KEATS-2), that helps the knowledge engineer build the actual expert system. Therefore, KEATS-2 allows the knowledge engineer to map the conceptual models, devised using Acquist, into frame structures at the knowledge representation level. As the different levels of analysis are more tightly integrated, we feel that the idea of knowledge engineering as iterative refinement of models is better supported in KEATS.

The approach taken by KRITON is slightly different from that offered by Acquist and KADS. KRITON aims to make knowledge acquisition fully automated, by eliciting, analyzing and representing knowledge. A number of knowledge elicitation techniques are prestored: interviews, incremental text analysis and protocol analysis. Interviews are carried out through the use of the repertory grid technique and aim to pull out declarative knowledge. Protocol analysis produces the goal structure of the domain, and text analysis provides the knowledge engineer with statistical information about the text. Such declarative and procedural knowledge is encoded in an intermediate knowledge representation formalism, that uses a propositional calculus and a language for describing devices. This intermediate representation is semi-automatically translated into a machine runnable representation, with the goal structure becoming a rule set, and the content structure a frame base. Other processes act on the knowledge base, by propagating constraints and checking for inconsistencies.

As KRITON also supports different levels of analysis, including the linguistic one, much of the philosophy of the system is similar to KADS and KEATS. The main difference between these approaches concern the fact that while KRITON aims to make the knowledge acquisition process fully automated, KADS and KEATS provide tools that support a more comprehensive and principled methodology for knowledge engineering. This can be seen by comparing the notion of 'intermediate level' in KRITON with that of domain conceptualization in KEATS. The former seems only instrumental to the way the system works, the latter is a fundamental part of knowledge engineering.

5. CONCLUSIONS

Our description characterizes knowledge acquisition as the elicitation and analysis of data in order to produce conceptual models of the target problem. We have also emphasized the linguistic nature of the data analysis activity. Knowledge engineers need tools that could help them progressing trough the knowledge acquisition stages to the implementation of the final expert system The limitations of KEATS-1 have led to the development of Acquist, a hypertext-style tool, with the provision of facilities to integrate a bottom-up style of textual analysis with a top-down (model-driven) approach. Acquist incorporates a graphical 'map' facility which helps to bridge the gap between a sketched-out account of overt behaviour and a true (runnable) internal representation thereof. We feel strongly that this gap-bridging is important if knowledge acquisition is to remain firmly in the AI tradition of providing testable (executable) theories.

6. REFERENCES

Anjewierden, A., Knowledge Acquisition Tools. *AI Communications*, Vol 0 No. 1, pp 29-38, 1987.

Bennet, J.S. ROGET: A knowledge-based consultant for acquiring the conceptual structure of an expert system. Stanford Heuristic Programming Project, Memo HPP-83-24, Stanford University, October 1983.

Boose, J. A Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology. *Int. J. Man-Machine Studies*, 23, 1985.

Breuker, J., and Wielinga, B. KADS: Structured Knowledge Acquisition for Expert Systems. Proceedings of Expert Systems and their Applications, Avignon, 1985.

Burton, A. M., Shadbolt, N. R., Hedgecock, A. P., and Rugg, G. A. formal evaluation of knowledge elicitation techniques for expert systems: domain 1. In D. S. Moralee(ed), *Research and Development in Expert Systems 4*, Cambridge University Press, 1987.

Chandrasekaran, B. Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks. Proceedings IJCAI 87, Milan, 1987.

Clancey, W. J. Heuristic Classification. *Artificial Intelligence* 27, 1985.

Clancey, W. J. The Epistemology of a Rule-Based Expert System - a Framework for Explanation. Artificial Intelligence 20, 1983.

Conklin, J., Hypertext: An Introduction and Survey. *IEEE Computer Magazine*, pp 17-41, September 1987.

Davis, R., Lenat, D. *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, 1982.

Diederich, J., Ruhmann I., and May, M. KRITON: a knowledge-acquisition tool for expert systems. *Int. J. Man-Machine Studies*, 26, 1987.

Ericsson, K. A., Simon, H. A. Protocol Analysis. Verbal reports as data. Cambridge, MA, 1984.

Eshelman, L., McDermott, J. MOLE: A Knowledge Acquisition Tool That Uses Its Head. *Proceedings AAAI 86*, Philadelphia, PA, 1986.

Feigenbaum, E. A. The Art of Artificial Intelligence: Themes and case studies of knowledge engineering. *Proceedings IJCAI 77*, Cambridge MA, 1977.

Goodman,.D. The Complete HyperCard Handbook, Bantom Books, 1987.

Hayes, P. J. In defence of Logic, *Proceedings IJCAI 77*. Los Angeles: Morgan Kaufmann, 1977.

Hayes-Roth, F., Waterman, D. A., and Lenat, D. B. *Building expert systems*, Reading, Massachussetts: Addison-Wesley 1983.

Johnson, S. C., Hierarchical clustering schemes. Psychometrika, 32, 1967.

Klinker, G., Boyd, C., Genetet, G., and McDermott, J. A KNACK for Knowledge Acquisition. *Proceedings AAAI 87*, Seattle, WA, 1987.

Mason, R. J., and Rose, S. P. R. Passive avoidance learning produces focal elevation of bursting activity in the chick brain: amnesia abolishes the increase. *Behavioural and neural biology*, 1988.

Motta, E., Eisenstadt, M., Pitman, K., and West, M. Knowledge Acquisition in KEATS: The Knowledge Engineer's Assistant. *Expert Systems*, 5(2), 1988 (Also available as Technical Report No. 30, Human Cognition Research Laboratory, The Open University, Milton Keynes, U.K., 1988.)

Olson, J. R. and Reuter, H. H. Extracting expertise from experts: Methods for Knowledge Acquisition, *Expert Systems* 4(3), August 1987.

Pitman, K. CREF: An Editing Facility for Managing Structured Text. Technical Report No. 14, Human Cognition Research Laboratory, The Open University, Milton Keynes, UK. 1985.

Shepard, R. N., Romney, A. K., and Nerlove, S. B. Multidimensional Scaling: Theory and Applications in the Behavioural Sciences, Volume 1, Seminar Press, New York, 1972.

Stallman, R.M. EMACS - The Extensible, Customizable, Self Documenting Display Editor, Memo No. 519, Artificial Intelligence Laboratory, MIT, Cambridge massachusetts, June 1979.

Wielinga, B. J., and Breuker, J. A. Models of expertise. *Proceedings ECAI 86*, Brighton, U.K., 1986.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Developing a Tool for Knowledge Integration: Initial Results[1]

Kenneth S. Murray (murray@cs.utexas.edu)
Bruce W. Porter (porter@cs.utexas.edu)

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

Abstract

Knowledge integration is the task of incorporating new information into existing knowledge. The task is difficult because the consequences of an addition to an extensive knowledge base can be numerous and subtle. Current methods for automated knowledge acquisition largely ignore this task, although it is increasingly important with the move toward large scale, multifunctional knowledge bases. To study knowledge integration, we propose to develop and evaluate a knowledge acquisition tool that helps with extending a knowledge base through interaction with a knowledge engineer. An initial prototype of this tool has been implemented and demonstrated on a complex extension to a large knowledge base.

1. Introduction

Knowledge integration is the task of incorporating new information into a knowledge base. It requires elaborating new information and resolving inconsistencies with existing knowledge. The complexity of knowledge integration is due to the numerous, subtle interactions between new information and existing knowledge.

The importance of researching knowledge integration has become apparent to us while building a multifunctional knowledge base in the domain of botany. The complexity of adding new information grows with the size of the knowledge base. For example, adding new information that carbon dioxide is a raw material in photosynthesis has significant consequences for the existing knowledge. Some consequences identify missing information, such as the mechanism for acquiring the raw material. Others reveal anomalies in the knowledge base, such as the conclusion that plant embryos in seeds must die because they lack carbon dioxide. Determining these consequences is hard but results in acquiring important domain knowledge.

Knowledge integration is crucial to automated knowledge acquisition, although it has received little attention. A tenet of current knowledge acquisition research is that

[1] Support for this research was provided by the Army Research Office under grant ARO-DAAG29-84-K-0060 and the National Science Foundation under grant IRI-8620052.

knowledge base refinement occurs (primarily) in the context of repairing a problem solving failure. Current tools acquire knowledge to repair the failure, but ignore the consequences of the new information on the existing knowledge base. The task of knowledge integration is to determine these consequences.

To study knowledge integration, we are developing a learning tool, KI, which plays an active role in extending a knowledge base. When new information is provided, KI uses relevant domain expertise already encoded in the knowledge base to elaborate the new information and to determine its relationship to existing knowledge. KI elicits information from the knowledge engineer to fill the gaps and resolve the anomalies revealed in the knowledge integration process.

The "laboratory" for this research is the Botany knowledge base [PORT88] which contains task-independent knowledge about plant anatomy, physiology, and development. Using the excellent tools of the CYC project [LENA87], our group has been constructing the knowledge base for eighteen months; it currently contains 4000 frames.

In this paper we discuss the increasing importance of research on knowledge integration and our initial results. In Section 2, we describe the trend toward multifunctional knowledge bases and the difficulties it poses for current knowledge acquisition methods. In Section 3, we describe our on-going research on knowledge integration with an example of extending the Botany knowledge base. In Section 4, we discuss lessons learned and future directions.

2. Some Future Requirements for Knowledge Acquisition Tools

The inflexibility of knowledge based systems is crucial to current methods for automated knowledge acquisition. Each system is committed to a particular problem solving method and task, such as heuristic classification to identify a patient's illness. Knowledge acquisition tools exploit this commitment in various ways. Some tools acquire knowledge by instantiating the model of a particular problem solving method [ESHL87] or task [MUSE87, BENN85]. Others acquire knowledge by debugging the knowledge base when problem solving fails [DAVI77, BARE87]. Still others acquire knowledge by generalizing a set of examples to form classification rules [QUIN86, MICH80]. Exploiting the specificity of a "target" knowledge base is the primary source of power of current tools.

The trend toward multifunctional knowledge bases (MFKB) requires new methods for automated knowledge acquisition. An MFKB is not committed to any particular problem solving method or task. Rather, it encodes general knowledge of a domain that should support diverse methods and tasks. For example, an MFKB for human physiology might be the substrate for expert programs that diagnose patients, tutor

students, and organize reference materials. Automated tools for helping construct an MFKB cannot exploit knowledge base specificity in the conventional ways.

Knowledge acquisition tools for MFKBs must adopt a general model of learning. The premise of most current tools is that learning involves instantiating a general model and debugging the result. This rigidity is inappropriate for constructing MFKBs. The premise of knowledge integration is that learning involves relating new information to existing knowledge. This has two subtle, yet significant, ramifications for knowledge acquisition tools. First, existing knowledge determines what is learned, not some *a priori* model of a particular problem solving method or task. Second, knowledge is acquired when a gap or an anomaly is revealed while relating new information to existing knowledge. The challenge is controlling the search for gaps and anomalies without committing to a specific problem solving method or task.

Multifunctional knowledge bases are inevitable. The grid of potential knowledge bases has three dimensions: domain, task, and problem solving method. Building knowledge based systems for individual cells of this grid is both costly and short-sighted. Growing interest in MFKBs is evidenced by the first workshop on the topic [WORK88] and several projects, including CYC [LENA87], Fermi [LARK88], Kreme [ABRE86], and Botany [PORT88]. The next section describes our initial efforts at automating knowledge integration for constructing multifunctional knowledge bases.

3. KI: A Tool for Knowledge Integration

We are developing KI, a knowledge acquisition tool that helps a knowledge engineer to extend a knowledge base. When new information is provided, KI uses the existing knowledge to critique the new information and determine its consequences. Our computational model of knowledge integration includes three prominent activities:

1) Recognition: identifying the knowledge relevant to new information.

2) Elaboration: applying the expectations provided by relevant knowledge to determine the consequences of the new information.

3) Adaptation: modifying the knowledge base to accommodate the elaborated information.

In this section, we describe our initial progress on KI. In Section 3.1, we present an example of knowledge integration that is representative of the complex knowledge base extensions we expect KI to perform. ProtoKI is a prototype implementation of KI that has been successfully tested with this example. In sections 3.2 through 3.4, we describe how ProtoKI performs the tasks of recognition, elaboration, and adaptation.

3.1 An Example of Knowledge Integration

This example involves extending the Botany knowledge base with new information about plant seeds. The knowledge base already has information that plant seeds contain nutritive tissue called endosperm, which is analogous to an egg's yolk. The plant embryo consumes the endosperm during its development inside the seed. A knowledge engineer wishes to extend the knowledge base with a representation of *nonendospermic seeds*, a taxon for seeds that contain no endosperm. The task of ProtoKI is to interact with the knowledge engineer to integrate this new information.

The knowledge engineer presents new information to ProtoKI: [2]

Knowledge Engineer: "There is a class of seeds that have no endosperm."

ProtoKI retrieves knowledge structures relevant to this new information. Using inference rules defined for the retrieved knowledge structures, ProtoKI forms the expectation that the embryo in a nonendospermic seed will die of starvation. It then attempts to confirm this expectation with the knowledge engineer:

ProtoKI: "These seeds do not give rise to healthy seedlings."

Knowledge Engineer: "Explain."

ProtoKI: "The embryo starves since the endosperm provides nutrients, and nutrients are essential for health and survival."

Knowledge Engineer: "Yes, nutrients are essential; however, the embryo survives."

ProtoKI attempts to resolve the inconsistency between its expectations and the assertion that embryos in nonendospermic seeds survive. One possible explanation is that these embryos acquire food from some other source. ProtoKI searches for alternate ways an embryo-stage plant might acquire food. A focused discussion with the knowledge engineer ensues that resolves the anomaly and extends the knowledge base.

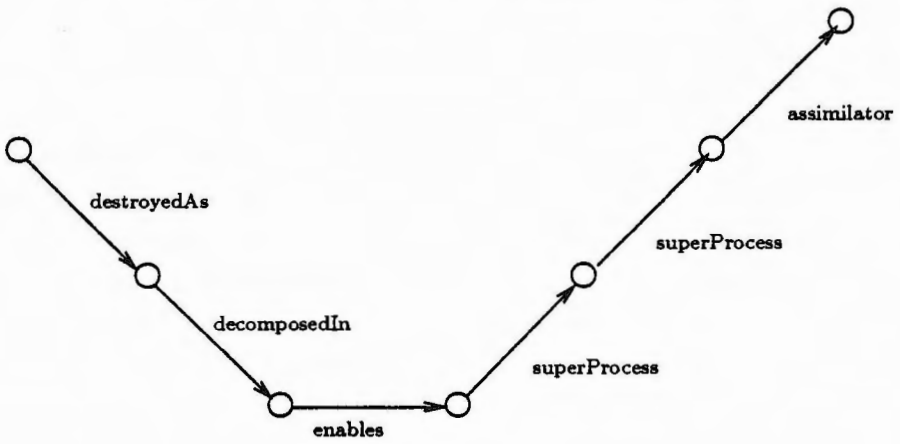
ProtoKI: "Does the embryo acquire nutrients from photosynthesis?"

Knowledge Engineer: "Yes, the shoots of some plant embryos emerge from the seed during germination and start producing photosynthate."

As this example illustrates, ProtoKI integrates new information by determining its consequences. When anomalies are encountered, ProtoKI suggests possible explanations to resolve them.

[2] This example has been simplified for presentation. For example, ProtoKI does not generate and parse natural language; this discourse has been converted from a language of frames, slots, and values.

Figure 1: The Perspective *Plant Food Source*



This semantic-net template defines the context relevant to an object in its role as a plant food source: A plant food source must have a stage when it is destroyed and decomposed into nutrients. This decomposition enables the nutrients to be assimilated by the plant. The nested subprocesses of assimilation are provision and attainment.

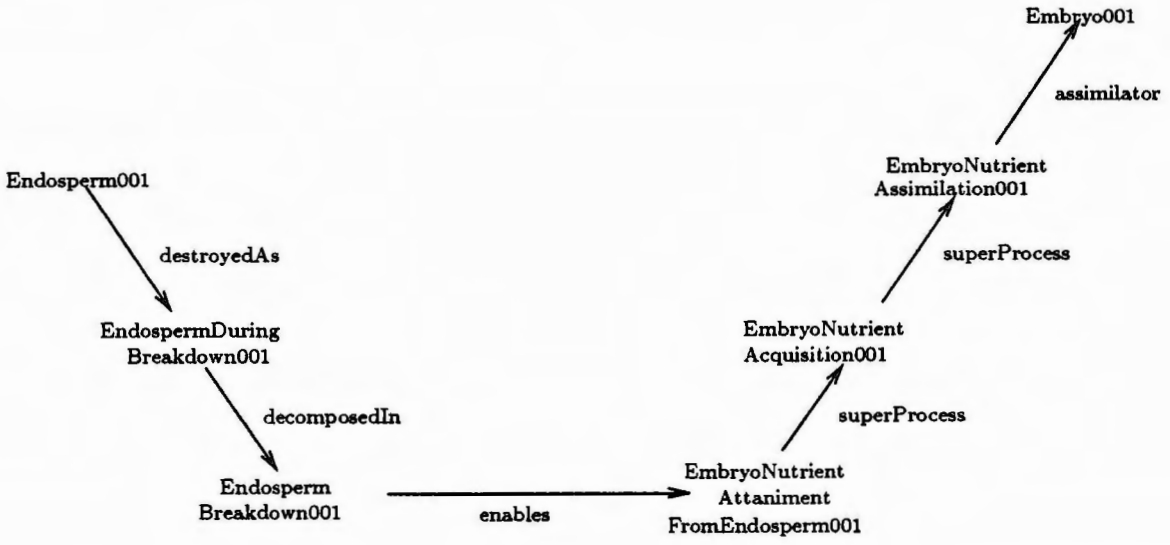
The computational issues that arise during knowledge integration include identifying knowledge relevant to new information, relating new information to the relevant knowledge, and adapting the knowledge base to accommodate the new information. The following three sections describe in greater detail how ProtoKI performs these activities.

3.2 Recognition

ProtoKI integrates new information by identifying relevant knowledge structures within the knowledge base. In the previous example about seeds with no endosperm, ProtoKI must determine which among the thousands of frames in the Botany knowledge base may be affected in some interesting way.

To focus the search for knowledge relevant to new information, the representation of each object in the Botany knowledge base is structured with *perspectives.* Each perspective is a partitioning of the knowledge base that describes an object in a particular role. For example, one perspective of endosperm is *Plant Food Source*, as shown in Figure 1. Other perspectives include: endosperm as a *Product Of Reproduction,* endosperm as a *Contained Object,* and endosperm as a *Taxon Defining Part.* ProtoKI collects the perspectives for objects referenced by new information and prompts the knowledge engineer to select which are appropriate.

Figure 2: The Context Created by Instantiating *Plant Food Source*



A perspective is a semantic-net template that can be instantiated for hypothetical objects. The instantiation of *Plant Food Source* for an endosperm is presented in Figure 2. Collectively, these instantiated frames comprise a *context* representing an endosperm in its role as a plant food source; this context is used to simulate the effects of the new information about endosperm.

3.3 Elaboration

During recognition, ProtoKI has isolated the context in the knowledge base most relevant to the new information. Next, during elaboration, ProtoKI determines how the new information interacts with existing knowledge within this context. Elaboration involves applying inference rules to propagate the effects of the new information throughout the context.

In the endosperm example, elaboration begins when ProtoKI asserts that the endosperm is absent from the context by assigning value *False* to the slot *enabled?* of *Endosperm001*. This assignment triggers inference rules that determine the consequences of seeds lacking endosperm. For example, without the endosperm, the embryo cannot get enough nutrients to survive. The inference rules applicable to this example are listed in Figure 3.

Through elaboration, ProtoKI concludes that the plant embryo is dying from lack of nutrients. This triggers the instantiation of a second semantic-net template defined for plants that are starving and in danger of dying. The context is expanded to include the plant's developmental stages immediately before and after its embryo

1. When an entity is disabled, all of its developmental stages are disabled.
2. When an entity is disabled, all the processes involving the entity are disabled.
3. When a process is disabled, all the processes that its completion enables are disabled.
4. When the known methods of attaining some essential resource is disabled, the rate of acquiring the resource is inadequate for survival.
5. When the assimilation rate for some resource is unknown, it is the same as the acquisition rate.
6. When nutrient assimilation is inadequate for survival, the assimilator is dying.
7. When a living entity is dying, the succeeding developmental stage is disabled.

Figure 3: Heuristic Rules Relevant to Endosperm as a Plant Food Source

stage and how nutrients are acquired during each of these developmental stages. Through continued elaboration, ProtoKI concludes that the plant's seedling stage is not reached because the plant dies during its embryo stage. The extended context is presented in Figure 4.

An important function of elaboration is identifying confounded expectations. These occur when expectations of the knowledge base are violated by new information or when two rules reach conflicting conclusions. Resolving inconsistencies involves correcting the new information to comply with current expectations or adapting the existing knowledge structures to accommodate the new information.

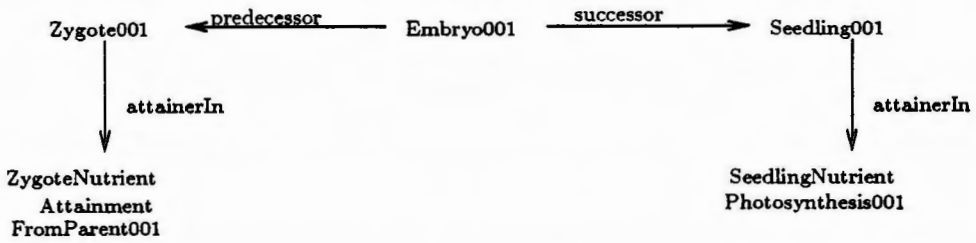
3.4 Adaptation

Elaboration can reveal anomalies in the knowledge base and the adaptation step resolves them. An anomaly can result from inconsistencies introduced by either inference rules used during elaboration or assertions entered by the knowledge engineer. In the endosperm example, an anomaly is detected when the knowledge engineer asserts that the embryos of nonendospermic seeds survive, correcting the prediction that these embryos starve.

Resolving anomalies requires correcting explanations that support failed expectations and constructing alternative explanations to account for new information. When the prediction that embryos of nonendospermic seeds starve is refuted by the knowledge engineer, the explanation for this prediction is inspected to determine its weakest premise. Rule 4 (from Figure 3) relies on a closed-world assumption and is considered a relatively weak inference. Therefore, its conclusion is retracted, and *Embryo Nutrient Acquisition 001* is assumed to be adequate for the embryo's survival. This change propagates through the explanation, retracting the belief that the embryo starves.

The original anomaly has been resolved by assuming adequate nutrient acquisition by the embryos of nonendospermic seeds. However, no alternative method is known

Figure 4: The Extended Context



This is the extension to the context of Figure 2, showing the developmental predecessor and successor of Embryo001 and their methods of nutrient attainment.

for plant embryos to acquire nutrients. ProtoKI seeks to construct an explanation for the assumed nutrient acquisition using the following inference:

If a resource acquisition is adequate for survival, but no attainment method is known, then assume the attainment method of the developmental successor is employed.

This rule suggests the embryos of nonendospermic seeds attain sugar by photosynthesis, as is done by seedlings. However, this hypothesis introduces new constraints on the embryos of nonendospermic seeds. For example, to attain nutrients by photosynthesis, the embryo must be a photosynthetic plant. Therefore, to apply this inference, ProtoKI asserts that *Embryo 001* is an instance of *Photosynthetic Plant*. As a photosynthetic plant, the embryo inherits the following features: its color is green, it contacts sunlight, and its composition includes chlorophyll. In short, the plausibility of explaining the survival of nonendospermic embryos by assuming they engage in photosynthesis is contingent on their contacting sunlight and possessing chlorophyll. Confirming these assumptions leads to the acquisition of further knowledge from the knowledge engineer.

4. Assessment of the Initial Results

The preceding example is representative of the complex knowledge base extensions that a knowledge integration tool should handle. Implementing this scenario has provided several insights into the task of knowledge integration and our approach to automating it. In this section, we highlight the lessons derived from this exercise.

4.1 Source of Power

KI uses the domain expertise of the knowledge base to determine the consequences of new information. By contrast, other approaches to critiquing knowledge base extensions under-utilize the existing knowledge. These include:

- Teiresias [DAVI77] critiques new rules with *rule models.* Rule models record correlations between antecedent terms and consequent terms of the existing rule base. If a new rule violates a rule model, Teiresias suggests how the rule can be modified to conform to the model.

- Knac [LEFK87] anticipates knowledge base revisions with discourse cues and a heuristic theory of knowledge acquisition. Knac is capable of predicting which structures to revise when new information is ambiguous.

- Odysseus [WILK88] validates new rules with a *confirmation theory.* This involves techniques to assess the quality of a new rule for subsequent problem solving, and to ensure its consistency against a set of test cases.

Each of these methods demonstrates the use of external knowledge or meta-knowledge to evaluate new information. However, they cannot detect subtle anomalies introduced by knowledge base revisions because they ignore the consequences of new information for existing domain knowledge. FIE [COHE84] uses stylized resolution to determine the "shallow" interaction between new information and existing beliefs. However, this approach lacks sufficient control to integrate extensions into a large knowledge base or to identify the "deep" consequences of new information.

4.2 Source of Complexity

The first step of knowledge integration requires identifying existing knowledge that might be effected by new information. This task is quite difficult because of four complexities. First, anomalies introduced by knowledge base revisions are often implicit. This requires exploring how new information is propagated by the system's inference rules. However, computing the deductive closure of a knowledge base is intractable. Second, the paths through the network relating two frames may be quite long. Since these may lead to the "deep" consequences of the new information, the paths must be pursued. Third, the frames of the knowledge base are densely interconnected. This precludes unfocused search methods, such as spreading activation. Finally, since no fixed performance task is assumed, we cannot simply consider the consequences of new information against a fixed set of queries.

4.3 Limitations

Our implementation of a tool for knowledge integration has made several simplifying assumptions. For example, the task of controlling the selection of perspectives was trivialized. We are designing an agenda-based architecture for the next phase of development. Tasks on the agenda correspond to selecting and instantiating perspectives. A perspective is selected for possible instantiation when its degree of overlap with the current context exceeds an activation threshold. Selected perspectives are ordered for instantiation by a heuristic measure of "interestingness." This is a function of the conflict level of anomalies contained in those portions of the context that

each perspective overlaps. This control mechanism permits KI to conduct a best-first search through the deductive closure of the extended knowledge base, either autonomously or under the user's guidance.

Because knowledge integration assumes substantial domain knowledge, tools for knowledge integration are limited. In particular, they are inappropriate during the initial stages of knowledge base development when the encoded domain expertise is sparse.

5. Summary

Knowledge integration is the task of incorporating new information into a knowledge base. This is an important and difficult task because the interactions between new information and existing knowledge can be numerous and subtle. Current knowledge acquisition tools ignore these interactions.

Knowledge integration requires an active learner following three steps: recognition, elaboration, and adaptation. Given new information to incorporate into a knowledge base, recognition identifies known concepts that relate to the new information. Elaboration establishes these relationships while filling-in missing details of the new information and identifying conflicts with existing knowledge. Adaptation modifies the learner's knowledge to accommodate the elaborated information.

Our approach to knowledge integration recognizes the inevitable trend toward multi-functional knowledge bases. A multi-functional knowledge base encodes general knowledge of a domain that should support diverse tasks and problem solving methods. Current knowledge acquisition tools are inappropriate because they exploit the commitment of a knowledge base to a particular task and method.

We are testing our approach to knowledge integration with a tool that helps with the arduous task of extending a multi-functional knowledge base. The tool acquires knowledge when a gap or anomaly is revealed while relating new information to existing knowledge. An initial prototype of this tool has been implemented and tested on the Botany knowledge base.

Acknowledgements

We are grateful to Rob Holte, Ray Mooney, Brad Blumenthal, and Karl Branting for reviewing early drafts of this paper. We appreciate the work of Karen Pittman, Art Souther, James Lester, Liane Acker, and Tom Jones in helping with the construction of the Botany knowledge base. Finally, we are indebted to Doug Lenat and the CYC project for inspiration and software, and to MCC for donating our computer equipment.

References

[Abre86] Abrett, G. and Burstein, M.H. The KREME Knowledge Editing Environment. *Proceedings of the Knowledge Acquisition for Knowledge-based Systems Workshop* (1986).

[Bare87] Bareiss, E.R., Porter, B.W. and Wier, C. Protos: An Exemplar-based Learning Apprentice. In *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufmann, 1987.

[Benn85] Bennett, J.S. ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. *Automated Reasoning 1:1* (1985), 49–74.

[Cohe84] Cohen, D. A Forward Inference Engine to Aid in Understanding Specifications. *Proceedings of the National Conference on Artificial Intelligence Intelligence* (1984), 56–60.

[Davi77] Davis, R. Interactive Transfer of Expertise: Acquisition of New Inference Rules. *Proceedings of the International Joint Conference on Artificial Intelligence* (1977), 321–328.

[Eshl87] Eshelman, L., Ehret, D., McDermott, J. and Tan, M. MOLE: A Tenacious Knowledge Acquisition Tool. *International Journal of Man-Machine Studies 26*, 1 (1987), 41–54.

[Lark88] Larkin, J.H., Reif, F., Carbonell, J. and Gugliotta, A. FERMI: A Flexible Expert Reasoner with Multi-Domain Inferencing. *Cognitive Science 12* (1988), 101–138.

[Lefk87] Lefkowitz, L.S. and Lesser, V.R. Knowledge Acquisition as Knowledge Assimilation. *Proceedings of the 2nd Knowlege Acquisition for Knowledge-Based Systems Workshop* (1987), 14.0–14.13.

[Lena87] Lenat, D.B. and Feigenbaum, E. On the Thresholds of Knowledge. *Proceedings of the International Joint Conference on Artificial Intelligence* (1987), 1173–1182.

[Mich80] Michalski, R.S. and Chilausky, R.L. Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from Examples: a Case Study Involving Soybean Pathology. *International Journal of Man-Machine Studies 12* (1980), 63–87.

[MUSE87] 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.

[PORT88] Porter, B., Lester, J., Murray, K., Pittman, K., Souther, A., Acker, L., and Jones, T. AI Research in the Context of a Large-scale Structured Knowledge Base. (Forthcoming Technical Report from the Department of Computer Science, University of Texas at Austin).

[QUIN86] Quinlan, J.R. Induction of Decision Trees. *Machine Learning 1,1* (1986).

[WILK88] Wilkins, D.C. Knowledge Base Refinement Using Apprenticeship Learning Techniques. *Proceedings of the National Conference on Artificial Intelligence* (1988), 646–651.

[WORK88] AAAI Workshop on Very Large Multifunctional Knowledge Bases. (Held at the National Conference on Artificial Intelligence, St. Paul, MN, August 22, 1988. Organizing Committee: Carbonell, J. Rich, E., Thomason, R., Nirenburg, S. and Monarch, I.).

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

An Editor for the Conceptual Models of Interactive Knowledge-Acquisition Tools

Mark A. Musen
Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University
Stanford, California 94305-5479
MUSEN@SUMEX-AIM.STANFORD.EDU

Computer-based tools can help system builders to create knowledge bases for expert systems. An important class of such knowledge-acquisition tools comprises programs that contain detailed models of families of application tasks. Users enter knowledge about particular applications into these tools in terms of the predefined task models that the tools incorporate. (For example, the OPAL knowledge-acquisition program uses a model of the task of administering cancer therapy to provide a framework that allows cancer specialists to enter the knowledge that defines individual cancer-treatment plans.) This paper describes PROTÉGÉ, an interactive program that assists knowledge engineers in the construction of task models, and that automatically generates custom-tailored knowledge-acquisition tools based on those models. Domain experts independently use the graphical tools that PROTÉGÉ generates to enter the knowledge that defines individual applications. The methodology separates the problems of creating models of application tasks from the entry of content knowledge, expediting the development of expert systems when multiple knowledge bases are required for related tasks.

1. Introduction

Knowledge acquisition, the process of eliciting the expertise of authorities in an application area so that computer systems can render domain-specific advice, is a problem in modeling. When the knowledge engineers who build expert systems interview experts in an application domain, the knowledge engineers form mental models of how the experts solve problems; the experts, of course, have mental models of their own that denote the problem-solving behavior associated with their profession. In the course of building a knowledge-based system, both the system builders and the experts continually revise their respective models. Although the knowledge engineers and the domain experts may have different mental models at the outset of their collaboration, the models eventually reach a convergence. This convergence is possible because the knowledge-acquisition process forces all parties to commit their mental models to a fixed, publicly examinable form—typically the emerging knowledge base. The often-cited difficulties of knowledge acquisition can, in general, be ascribed to the difficulties of creating and agreeing on this shared model of problem solving (Regoczei & Plantiga, 1987).

In many application areas, there are related tasks for which expert systems might be useful, and for which only one general task model is required. Once that model is available, system builders can extend the model to describe particular application tasks; each new knowledge base can then be viewed as a direct instantiation of the general task model. For example, the literature describes expert systems for troubleshooting electronic instruments, in which distinct knowledge bases are used for different devices (Freiling & Alexander, 1984); for advising on the use of data-analysis programs, in which separate knowledge bases are consulted for different statistical routines (Gale, 1987); and for administering cancer chemotherapy, in

which individual knowledge bases represent different oncology treatment plans (Musen et al., 1987). In such domains, the major obstacle to knowledge acquisition tends to be the initial formulation of the general task model, rather than the construction of the various knowledge bases that represent instantiations of that model (Musen, 1988a). Nevertheless, building the individual knowledge bases using traditional knowledge-engineering techniques is often a time-consuming undertaking.

Given a general, reusable task model, computer-based tools can greatly assist domain specialists in the development of new expert-system applications. OPAL (Musen et al., 1987; Musen et al., 1988), for example, is an interactive program that acquires new cancer-treatment plans for an expert system called ONCOCIN (Shortliffe et al., 1981)—a program that provides therapy advice to physicians who take care of cancer patients. Expert oncologists use OPAL to describe new chemotherapy regimens for ONCOCIN by filling out graphical forms and by drawing flowchart diagrams on a workstation display (Figure 1). The terms and relationships of a predefined model of cancer therapy frame all entries into the system. Consequently, users view the knowledge that they enter into OPAL in terms of this cancer-therapy model, rather than in terms of the underlying representations required by the ONCOCIN inference engine. The task model that physicians instantiate to create new knowledge bases establishes the semantics of the users' entries into OPAL, and thus forms the *conceptual model* that shapes all interactions with the program (Musen et al., 1987; Musen 1988a).

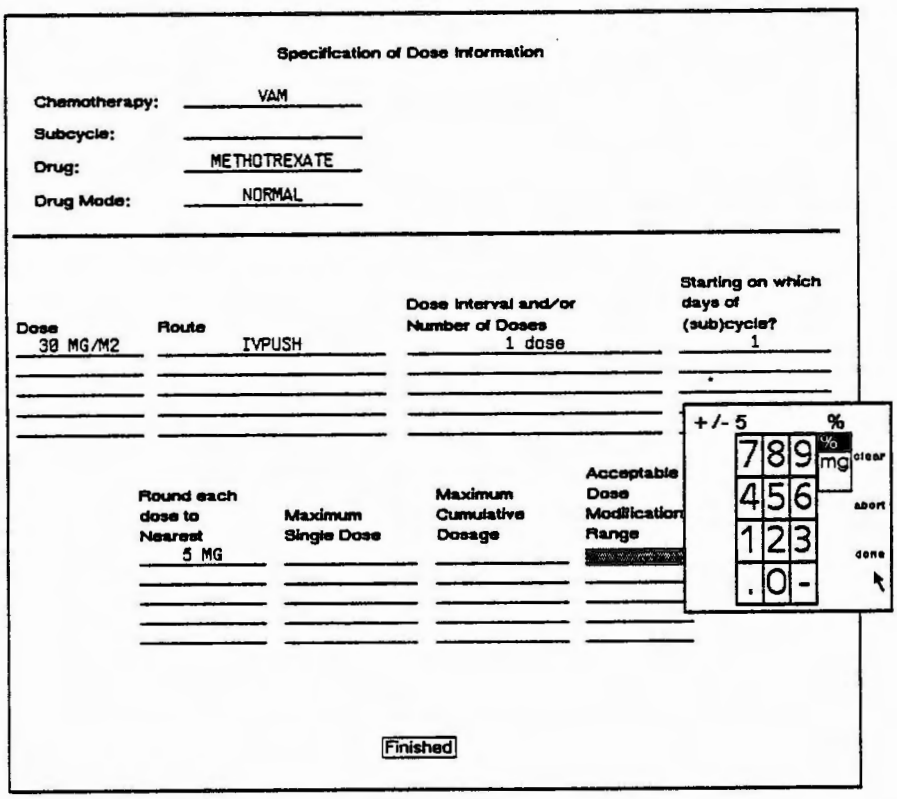


Figure 1: OPAL form for drugs. This graphical form allows physicians to fill in blanks to designate values for the attributes of chemotherapeutic drugs. The software input device at the right side of the form is known as a *register.*

Task-specific knowledge-acquisition tools such as OPAL can streamline the development of multiple, related knowledge bases. Building such tools is itself, however, a laborious undertaking. For example, OPAL required 3.5 person-years of work before an acceptable prototype was constructed, and updating the system occasionally demands considerable reprogramming. Yet just as computer programs can accelerate knowledge acquisition by capitalizing on similarities among classes of domain tasks, *metalevel* programs can accelerate development of systems such as OPAL by capitalizing on similarities among *classes* of knowledge-acquisition tools.

This paper describes PROTÉGÉ, a metalevel program that helps domain experts and knowledge engineers to create graphical tools like OPAL that are custom-tailored for new application areas (Figure 2). Using PROTÉGÉ, system builders can develop new domain-specific knowledge-acquisition tools in a few days. Before it is possible to discuss PROTÉGÉ in detail, however, it will be necessary to review the principles underlying OPAL. It is important to understand the conceptual model in OPAL, because it is such a model that knowledge engineers enter and edit explicitly using PROTÉGÉ.

2. OPAL: A Task-Based Knowledge-Editing Tool

Cancer-treatment plans (often referred to as *protocols)* are sufficiently alike that it has been possible to develop a model of the components that typically are present (Musen et al., 1987). This model comprises a set of terms and relationships by which a physician can specify the *task* of administering cancer therapy to a given patient enrolled in a given treatment protocol at a given point in time. For example, the terms of the model allow definition of the chemotherapies (drug combinations) that should be administered and the treatment modifications that should be considered. However, the cancer-therapy task model says nothing about the steps by which an oncologist actually arrives at a treatment recommendation, as there is no explicit notion of the problem-solving *methods* that are required. The cancer-therapy model merely addresses the structure of the task itself.

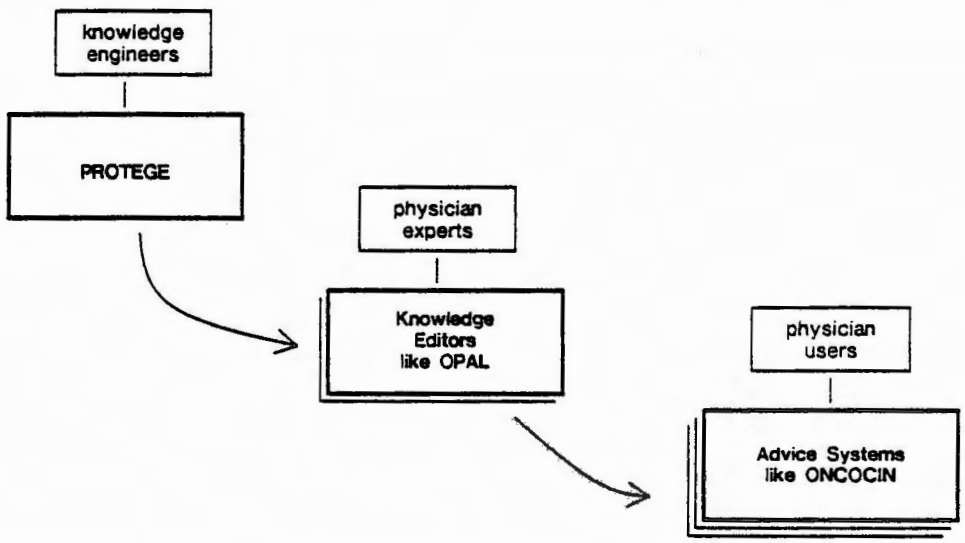


Figure 2: PROTÉGÉ—An architecture for generating advice systems. PROTÉGÉ generates graphical, task-based knowledge-acquisition tools. These tools, in turn, are used by expert physicians to create new knowledge bases for a domain-independent version of ONCOCIN, called *e-ONCOCIN.*

The model that physicians instantiate using OPAL has three main divisions. The first division delineates the *planning entities* in an oncology protocol (for example, the notions of chemotherapies and drugs), the relationships among those entities (for example, that chemotherapies are composed of drugs), and the entities' attributes (for example, that drugs have doses and routes of administration). The second division of the model identifies general *task-level actions* that can alter the administration of the planning entities (for example, the concept of delaying a chemotherapy or attenuating the dose of a drug). The third division of OPAL's model denotes the *input data* that can influence changes in therapy (for example, classes of side effects or laboratory tests). The three divisions of the task model are reflected in the graphical metaphors that OPAL uses to acquire knowledge from cancer specialists.

2.1 Planning Entities

The first division of the task model consists of an identification of the possible planning entities in the application area and the relationships among them. In the oncology domain, those entities are the therapeutic elements that constitute a treatment plan. The relationships among the entities are defined in part by a composition hierarchy that specifies how each element may be defined in terms of more specific components (Figure 3). Part of OPAL's function is to establish the specific treatment entities required in the particular cancer protocol being entered and to determine values for each entity's attributes.

The OPAL user interface consists primarily of a number of graphical *forms* that facilitate instantiation of the task model. For example, the OPAL form in Figure 1 allows the user to specify values for the attributes of a drug called *methotrexate* in the context of a chemotherapy called *VAM*. The program presumes that chemotherapies and drugs are entities in the domain and that they are related compositionally. When a user interacts with OPAL, he specifies values for items such as a drug's dose or route of administration by making selections either from predefined menus or from software input devices known as *registers*. OPAL displays the user's selection in the corresponding blank of the graphical form. Simultaneously, OPAL stores the entered value in a frame-based intermediate knowledge representation. Domain-specific programs subsequently translate OPAL's intermediate representation into knowledge bases for ONCOCIN. From the perspective of the user, however, knowledge entry is largely a matter of filling in the blanks of the various forms.

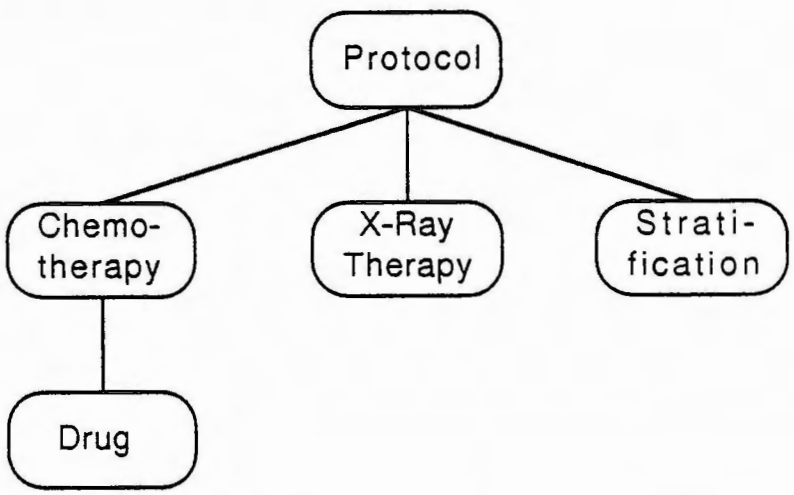


Figure 3: OPAL planning entities and their relationships. Treatment protocols in the oncology domain are composed of chemotherapies, X-ray therapies, and stratification steps. Chemotherapies are composed of drugs.

Although the values of most attributes of entities in OPAL can be specified using simple menus and registers, an important attribute of each oncology protocol is the high-level procedure (or *schema)* that defines the sequence of chemotherapies and radiation treatments that oncologists administer to their patients over time. The schema algorithms often are too elaborate to be entered via the simple fill-in-the-blanks approach used elsewhere in OPAL. Consequently, an iconic visual programming language has been developed to allow physicians to construct flowchart diagrams on the workstation screen to describe protocol schemas (Figure 4). By making selections from a palette of *reference icons* at the bottom of the display, users can add copies of these icons to a large rectangular region above the palette and then can position the icons using the mouse pointing device. By drawing links between the icons, oncologists can specify the flow of control from one element of the protocol to the next. The flowchart language is tailored for the oncology domain, yet allows users to specify general control notions such as sequential operations, conditional branches, iteration, concurrency, and exceptions using a graphical syntax (Musen et al., 1988). The schema flowcharts that physicians draw using OPAL are automatically converted to augmented transition networks that ONCOCIN can interpret.

2.2 Task-Level Actions

ONCOCIN's task of determining the appropriate treatment for a patient with cancer is solved using the method of skeletal-plan refinement (Friedland & Iwasaki, 1985). The hierarchical relationships among the planning entities (the chemotherapies, radiotherapies, stratification steps, and drugs) determine much of how that skeletal refinement proceeds. For example, initiating a plan for a new chemotherapy always requires invoking more detailed plans to administer the component drugs. Because the planning hierarchy is built into OPAL's model of the domain, the user does not have to specify these refinement actions explicitly when he enters new protocol knowledge; they are assumed to be constant from protocol to protocol.

On the other hand, there are several ways in which the direct instantiation of the skeletal plans can be modified. At the task level, oncologists can identify such behaviors as *attenuating* the customary dose of a drug, *substituting* one drug for another, or *aborting* or *delaying* chemotherapy. Although effecting one of these actions may require a number of different steps and may change ONCOCIN's treatment recommendation in a variety of ways, application experts tend to think about each of these task-level actions as a composite "chunk." Accordingly, OPAL

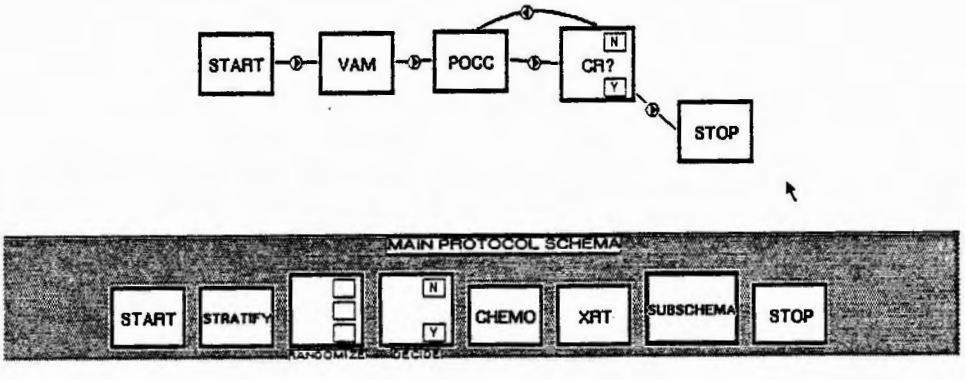


Figure 4: OPAL schema-entry environment. This flowchart diagram specifies a simple algorithm in which one cycle of VAM chemotherapy is administered, followed by cycles of POCC chemotherapy, until the ONCOCIN parameter *CR* (complete response to treatment) becomes true. Users constrict such flowcharts by joining together copies of icons selected from the palette beneath the graph. *XRT* stands for *X-ray therapy.*

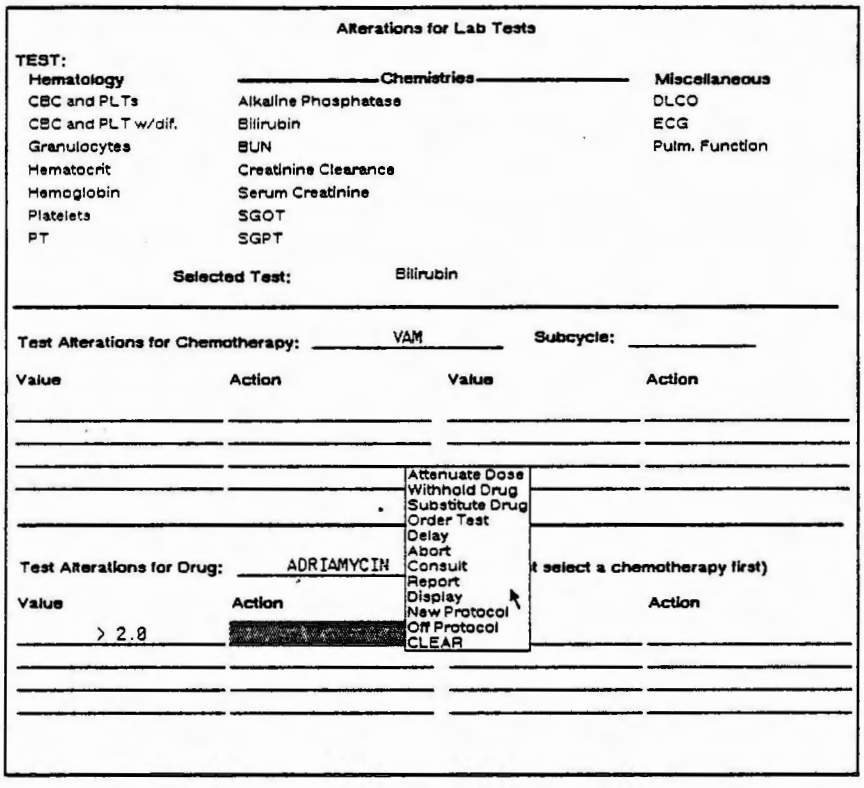


Figure 5: OPAL form for entering rules related to serum-chemistry results. The user is about to select a task-level action from the pop-up menu that will modify the treatment plan whenever a patient's serum-bilirubin value is greater than 2.0 mg/dl.

allows cancer specialists to select complex task-level actions directly from menus when they enter new protocols (Figure 5). The oncologists then can use additional graphical forms to enter attributes of the particular action, such as *by how much* to attenuate a drug or *how long* to delay treatment.

2.3 Input Data

Just as there is a limited set of task-level actions, there are limited classes of conditions that can predicate those actions. The results of *laboratory tests,* certain *patient conditions,* and manifestations of treatment-induced *toxicity* (side effects) are the principle factors that can dictate modification of the basic treatment plan in OPAL's domain.

Much of the knowledge needed to specify ONCOCIN's planning task consists of a mapping of specific input conditions onto corresponding task-level actions. OPAL simplifies entry of this portion of the knowledge by using a different graphical form for each class of conditions. For example, the form in Figure 5 displays the names of all the laboratory tests that have been predefined in OPAL's task model. When the user selects one of the laboratory tests using the mouse, she can enter and review rules predicated on the results of that test. In Figure 5, for example, the OPAL user is about to enter that, in the unusual circumstance that a patient's serum-bilirubin level is elevated to more than 2.0 mg/dl, the ONCOCIN user should consult with the person who wrote the protocol. The internal ONCOCIN rule that corresponds to the task-level specification is applicable only within a specific context—in this case, when giving a

particular drug (Adriamycin) in a particular chemotherapy (VAM) in a particular protocol. The left-hand side of the rule tests the value of the internal ONCOCIN parameter that represents the bilirubin value; the OPAL user, of course, is insulated from the actual rule syntax needed by the inference engine to perform the indicated comparison.

2.4. Current Status of OPAL

In 1986, 36 cancer protocols were encoded for ONCOCIN in rapid succession. The time required to enter the knowledge for a new protocol, formerly several weeks or months, was with OPAL a matter of days. OPAL continues to be used routinely at Stanford University to specify new oncology protocols and to edit the ONCOCIN knowledge base. Experience with OPAL suggests that, in certain domains, builders of expert systems can gain a great advantage by recasting the work of knowledge acquisition as the instantiation of predefined task models. Even though creating the initial OPAL model was difficult, the availability of the model for reuse has allowed expert oncologists who are unfamiliar with programming to create large knowledge bases on their own.

OPAL's task model can be viewed as a collection of terms and relationships. Suppose a knowledge engineer could create a *metalevel* model of the expected terms and relationships in a set of task models. The knowledge engineer could then write a computer program to assist in the instantiation of the metalevel model, defining new task models. This notion is the basis for development of PROTÉGÉ. Just as OPAL allows oncologists to generate ONCOCIN knowledge bases by instantiating a model of cancer protocols, PROTÉGÉ permits knowledge engineers to develop graphical knowledge-editing tools similar to OPAL directly by instantiation.

3. The PROTÉGÉ System

With PROTÉGÉ, a knowledge engineer defines a task model for a new domain (and hence defines the conceptual model for a new domain-specific knowledge-acquisition tool) by filling out a series of graphical forms in a manner similar to the way in which oncologists fill out the forms in OPAL. As in OPAL, the PROTÉGÉ forms cluster related data together for presentation to the user, and allow data to be examined and edited using direct-manipulation techniques. Whenever possible, users make choices from pop-up menus; keyboard entries are minimal.

The conceptual model presented by the forms in PROTÉGÉ mirrors the prototypical components of task models such as the one in OPAL; thus, the conceptual model in PROTÉGÉ is that of a class of models. PROTÉGÉ directly relates the task models that knowledge engineers enter into the system to ONCOCIN's problem-solving method of skeletal-plan refinement. Because PROTÉGÉ automatically maps the user's task model onto this problem-solving method, PROTÉGÉ's conceptual model also reflects a model of skeletal planning. The knowledge engineers who use PROTÉGÉ must learn to represent application tasks in terms of this skeletal-planning model. The model forms the basis for a graphical language that allows PROTÉGÉ's users to characterize different classes of application tasks that can be solved with a domain-independent version of ONCOCIN, called *e-ONCOCIN.*

The e-ONCOCIN program represents the current version of ONCOCIN without the latter's oncology-therapy knowledge base. All oncology-specific assumptions have been removed from the system's inference engine and user interface. The e-ONCOCIN shell, like the original ONCOCIN program, presumes that recommendations will be generated in the course of instantiating a frame hierarchy by invoking production rules and other methods associated with particular slots. The shell also incorporates ONCOCIN's time-oriented database system for storing the results of previous consultations that might bear on future planning actions. The assumptions that e-ONCOCIN makes about skeletal planning and about the discrete nature of temporal events cause the system to be particularly suited for generating advice in the general domain of clinical trials. Because it ultimately produces knowledge bases for e-ONCOCIN,

PROTÉGÉ is best viewed as a tool for creating task models (and, subsequently, special-purpose knowledge-acquisition systems) for clinical-trial applications (Musen, 1988b).

When PROTÉGÉ is first activated, the system's *main menu* appears on the workstation screen (Figure 6). This form provides access to other PROTÉGÉ forms that are available at the next organizational level. Via the main menu, the user can cause forms to be displayed for entering and editing the terms and relationships of a task model. Task models created using PROTÉGÉ have the same three divisions as the task model in OPAL: (1) planning entities, their relationships, and attributes (concepts such as *chemotherapy* and *drug*), (2) task-level actions (concepts such as attenuation of the dose of a drug), and (3) input data that can predicate those actions (concepts such as laboratory tests).

We shall now concentrate on how knowledge engineers enter and edit task models using PROTÉGÉ. To simplify the discussion and to provide familiar examples, PROTÉGÉ will be shown to generate portions of a knowledge-entry system for oncology protocols, called *p-OPAL*, that is similar to OPAL. More detailed information about both p-OPAL and a different PROTÉGÉ-generated

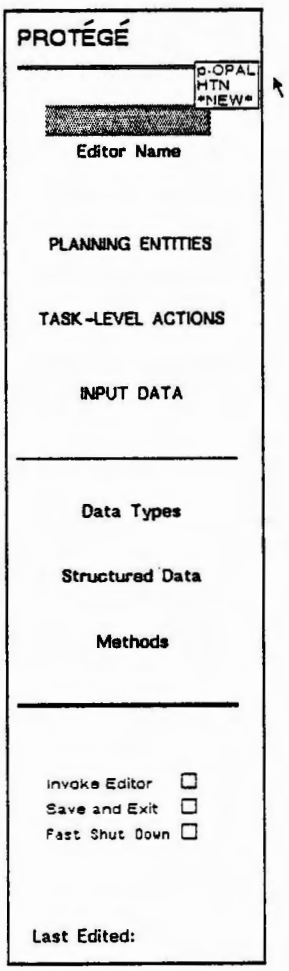


Figure 6: PROTÉGÉ main menu. The main-menu form asks the user for the name of the knowledge-acquisition system for which specifications are to be entered or edited via PROTÉGÉ. Once that name has been selected, users can access PROTÉGÉ forms for various topics by choosing the desired form from the list.

knowledge-acquisition tool that has been tailored for the domain of antihypertensive-drug therapy has been presented elsewhere (Musen, 1988b).

3.1 Planning Entities

At the core of every task model that a knowledge engineer enters into PROTÉGÉ is the notion of the skeletal plan that e-ONCOCIN instantiates during a consultation. In PROTÉGÉ, the plan as a whole is always called a *protocol*. PROTÉGÉ assumes that a protocol consists of a hierarchy of skeletal-plan components. Figure 7, for example, shows how a knowledge engineer has modeled the planning hierarchy in the oncology domain. The user has typed the names of the *classes* of planning entities into the column on the right side of the form. In the left column, he has indicated the name of the entity of which each entry in the right column is a component. (The entries in the left column are selected from pop-up menus that PROTÉGÉ constructs dynamically from the user's entries into the right column.) Thus, the specifications in Figure 7 indicate that stratification steps, chemotherapies, and X-ray therapies are all components of protocols in the oncology domain. Furthermore, chemotherapies are composed of drugs. Note that the PROTÉGÉ form says nothing about possible *instances* of these entities (for example, particular *kinds* of drugs); instance-specific details are always entered via the domain-dependent knowledge-acquisition tools that PROTÉGÉ generates automatically from users' task models.

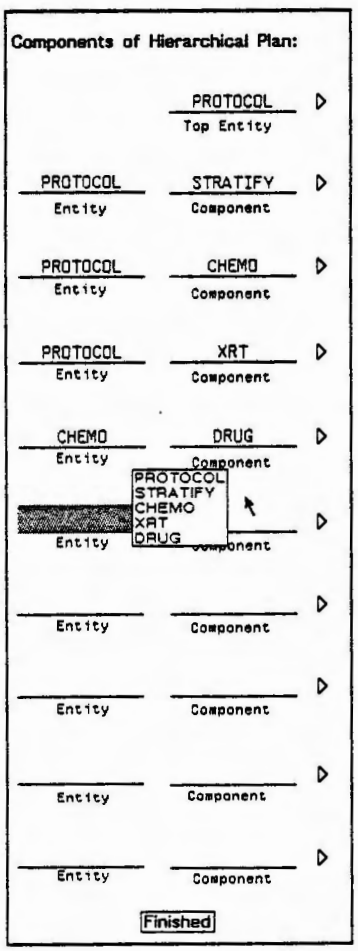


Figure 7: PROTÉGÉ form for planning entities. The knowledge engineer types in the names of the entities using the right column. Before he can type in the name, however, the user must identify the "parent" entity of which the new entity is a component using the left column. The specifications shown reflect the planning entities and relationships of the oncology domain depicted in Figure 3.

3.1.1. Procedural Attributes of Planning Entities

PROTÉGÉ uses the task description that a knowledge engineer enters into the system to create a task-specific knowledge-acquisition tool. Thus, knowledge of the planning entities in a new domain allows PROTÉGÉ to adapt the flowchart language used in OPAL (see Figure 4) to produce specialized flowchart languages. PROTÉGÉ assumes that each instance of an entity in the planning hierarchy (for example, a *protocol* instance) will comprise instances of its component entities (for example, instances of stratification steps, chemotherapies, and X-ray therapies) administered according to some time-oriented procedure. Thus, the data entered into the PROTÉGÉ form in Figure 7 are used to construct graphical environments in p-OPAL with which oncologists describe the sequence of operations that constitute a cancer-treatment protocol (Figure 8). PROTÉGÉ creates such a graphical environment for each class in the planning hierarchy that can be resolved into more specific time-ordered skeletal plans. Figure 9, for example, shows how p-OPAL allows its users to define the sequences of *drugs* that constitute particular *chemotherapies*. In both Figure 8 and Figure 9, the same *control* icons (START, STOP, DECIDE, RANDOMIZE, and SUBSCHEMA) appear in the palette beneath the region of the screen in which the flowchart is constructed. The additional icons that correspond to the relevant classes of planning entities, however, vary depending on the procedure to be defined. Thus, when p-OPAL is used to enter the algorithms for executing particular *protocols*, the palette incorporates special icons labeled STRATIFY, CHEMO, and XRT; when p-OPAL is used to enter algorithms for *chemotherapies*, the only special icon in the palette is labeled DRUG.

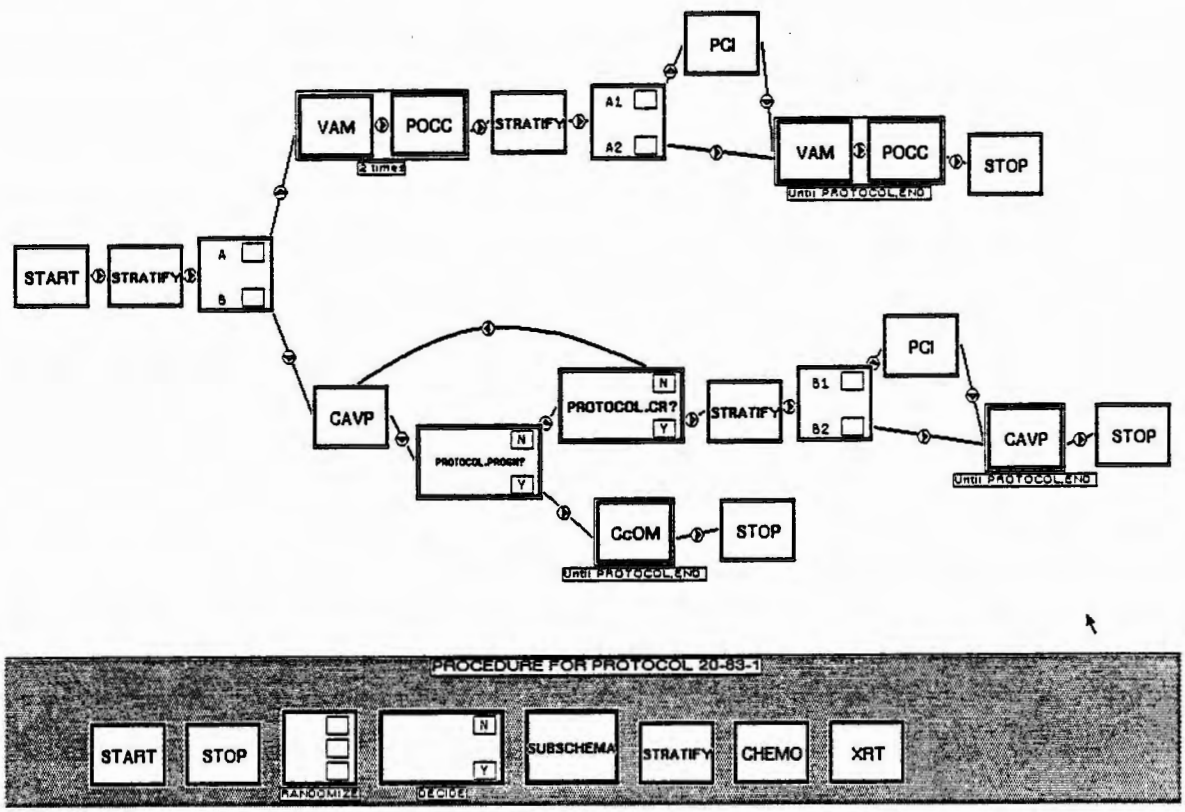


Figure 8: p-OPAL environment for entering the procedure associated with *protocol* entities. Compare this flowchart with the OPAL diagram in Figure 4.

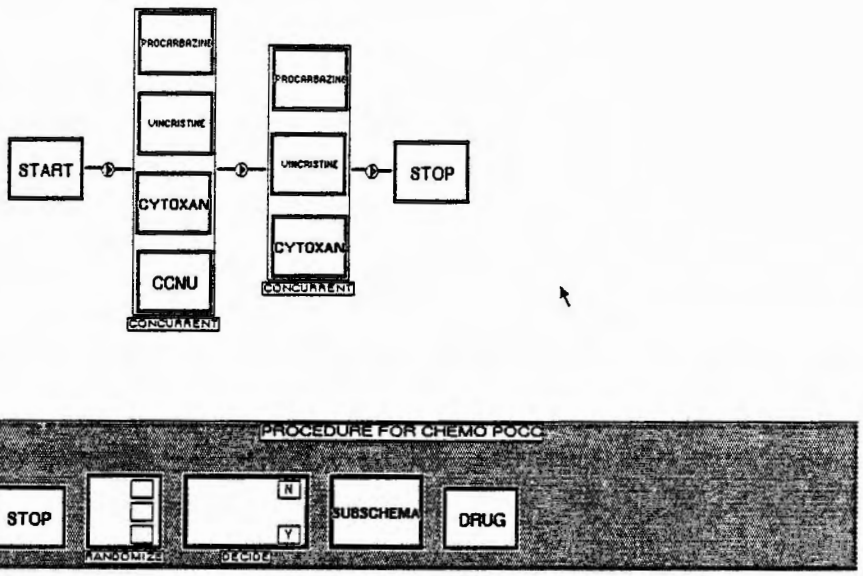


Figure 9: p-OPAL environment for entering the procedure associated with *chemotherapy* entities. This procedure calls for four drugs to be given simultaneously on one visit, followed by concurrent administration of three of the drugs.

3.1.2 Static Attributes of Planning Entities

In defining a task model at the PROTÉGÉ level, the user must enter the attributes associated with each class of planning entity. Like the slots in a frame, the attributes of a planning entity represent those properties that distinguish one instance of the entity from another. For example, all *protocols* defined using p-OPAL have a NAME attribute; the value of that attribute (that is, the name of the particular cancer-treatment plan) can differentiate one *protocol* instance from another.

Knowledge engineers use PROTÉGÉ to define the attributes of each planning-entity class. When the user selects with the mouse one of the arrows in the form in Figure 7, PROTÉGÉ displays the *entity-attributes* form on the workstation screen (Figure 10). This form asks the knowledge engineer to declare all the attributes of the designated entity. The PROTÉGÉ user simply types the required names into the blanks of the form. Selecting the arrow next to one of the attribute names causes PROTÉGÉ to open yet another form, which allows the knowledge engineer to enter detailed information about the indicated attribute.

For each attribute of each planning-entity class in a task model, the knowledge engineer must state how that attribute's value is determined. One of three options is possible:

1. The value of an attribute may be constant for all instances of the class under all circumstances. For example, if the *duration* of all protocols in an application area is modeled as an integer measured in *days,* the knowledge engineer's task model might state that all *protocol* entities have an attribute called DURATION.UNIT that always has the value *DAYS*. Such constant attributes are said to be *predefined,* and have values that are fixed by the knowledge engineer at the PROTÉGÉ level.

2. The value of a planning-entity attribute may vary from one instance of the planning entity to another, but may remain constant for any given instance. For example, the task model used to construct p-OPAL stipulates that the maximum duration of protocols is variable, but

that the maximum duration of any *individual* protocol never changes. *Protocol* entities therefore have an attribute called DURATION, the value of which is established with p-OPAL when users define specific protocol instances. Such attributes are said to have *editor-established* values, since the values are determined by the users of the knowledge-editing programs that PROTÉGÉ generates.

3. The value of a planning-entity attribute may vary depending on the circumstances and must be redetermined by the advice system (that is, by e-ONCOCIN) each time a consultation is run. For example, the dose of a particular drug may vary from visit to visit in accordance with the patient's condition and the presence of possible side effects; in p-OPAL, *drug* entities have an attribute called CURRENT.DOSE, the value of which is concluded by production rules each time that an e-ONCOCIN case is run. Such attributes are said to have *computed* values.

When an attribute of a planning entity has a *predefined* value, PROTÉGÉ allows the knowledge engineer to fill in a blank to provide the particular datum. When the user specifies that an attribute's value is *computed*, PROTÉGÉ provides forms that permit the knowledge engineer to enter the methods that e-ONCOCIN should use to conclude the value (for example, invoking production rules or asking the advice-system user). When an attribute's value is *editor established*, however, it is up to the PROTÉGÉ-generated knowledge-acquisition tool to ascertain the value.

The tools that PROTÉGÉ generates therefore include a graphical form for each planning entity that needs to be instantiated. These forms solicit values for each editor-established attribute of the relevant entity. Figure 11, for example, shows the p-OPAL form that oncologists fill out to describe new protocols. Each entry in the form corresponds to an attribute defined at the

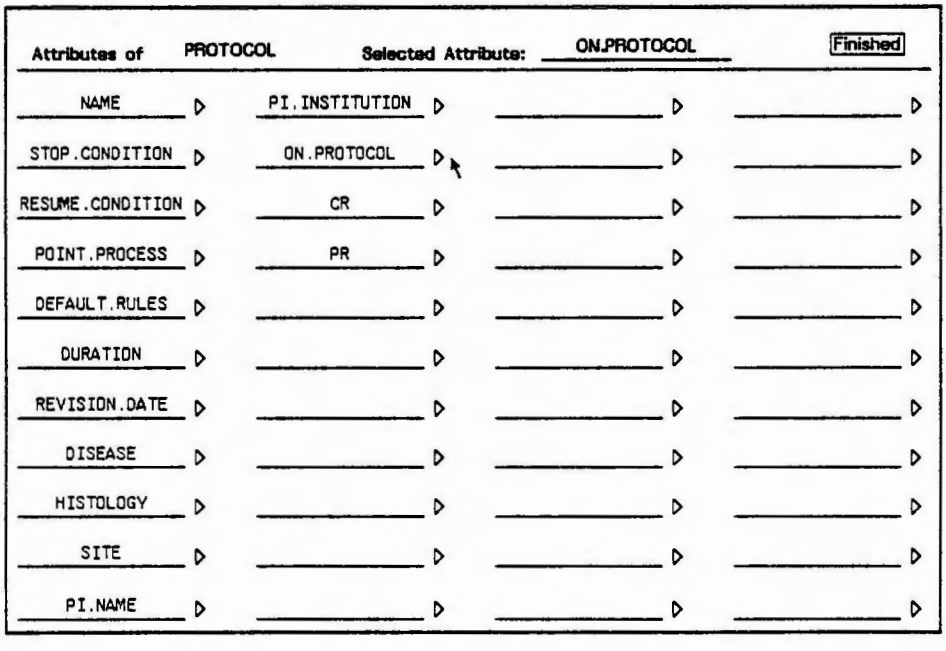


Figure 10: PROTÉGÉ form for planning-entity attributes. This form allows the knowledge engineer to list the attributes of a planning entity. Selecting one of the arrows to the right of an attribute name opens another form for describing the properties of the designated attribute. The attributes with editor-established values appear in the p-OPAL form in Figure 11.

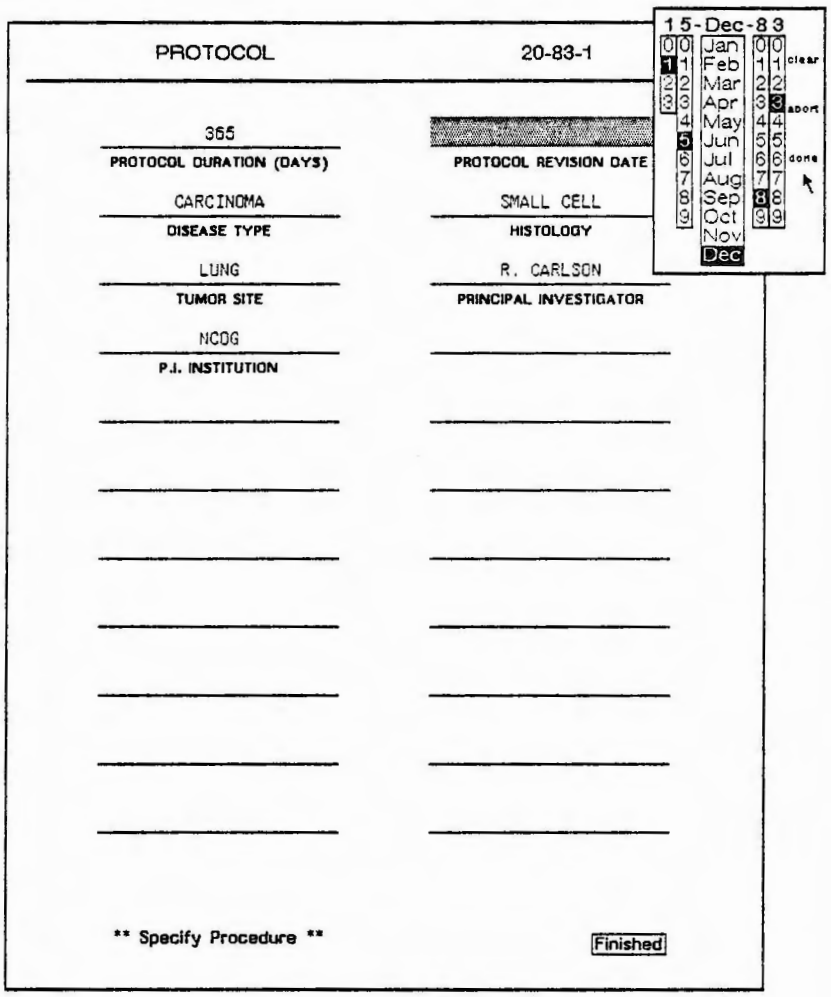


Figure 11: p-OPAL form for protocols. This form allows application specialists to enter values for all editor-established attributes of the protocol entity. Compare with the OPAL form in Figure 1.

PROTÉGÉ level (see Figure 10) that has an editor-established value. The labels for each blank of the p-OPAL form and the specialized menus and registers that acquire data from the p-OPAL user are all defined by knowledge engineers with PROTÉGÉ. The appearance and behavior of the p-OPAL form can thus be changed easily by editing the task model at the metalevel.

3.2 Task-Level Actions

The second division of a task model involves the *actions* that can modify instantiation of the basic skeletal plan. As with the definition of the planning entities, PROTÉGÉ provides special graphical forms that allow knowledge engineers to enter this part of a task model in domain-independent terms. By selecting "*Task-Level Actions*" from the PROTÉGÉ main menu (see Figure 6), the user causes the form in Figure 12 to appear on the workstation screen. The user then fills in the blanks to record the names of task-level actions in the domain being modeled. Additional forms then allow the knowledge engineer to describe how e-ONCOCIN should bring about each of the actions in the list.

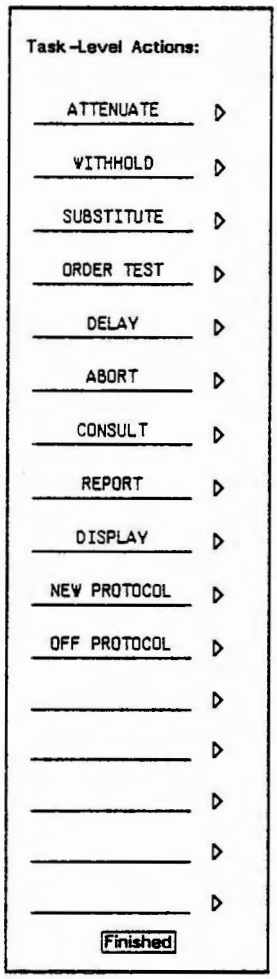


Figure 12: PROTÉGÉ form for task-level actions. This form allows knowledge engineers to list all the task-level actions in an application area.

PROTÉGÉ requires the knowledge engineer to define a procedure for executing each of the task-level actions in his model. The specific steps to be taken, however, often will be only partially specified at the PROTÉGÉ level, as the exact procedure may depend on parameters entered using the knowledge-acquisition tool that PROTÉGÉ generates from the task model. (For example, the number of days to delay chemotherapy is an attribute of the DELAY action that must always be entered into p-OPAL.) The procedure may also depend on parameters whose values are concluded by e-ONCOCIN at the time of a consultation. (For example, before the ATTENUATE action can be carried out, the dose of the drug given at the time of the last e-ONCOCIN consultation often must be known.) PROTÉGÉ asks the knowledge engineer to enter a partial plan (or *script)* for each task-level action in the application area, indicating where the task-level action depends on either editor-established or computed values (Figure 13). The script is written in terms of primitive operations such as "start execution of a planning entity" or "alter the value of an attribute of a planning entity." This simple language allows the knowledge engineer to define the semantics of each task-level action in a domain-independent manner (Musen, 1988b).

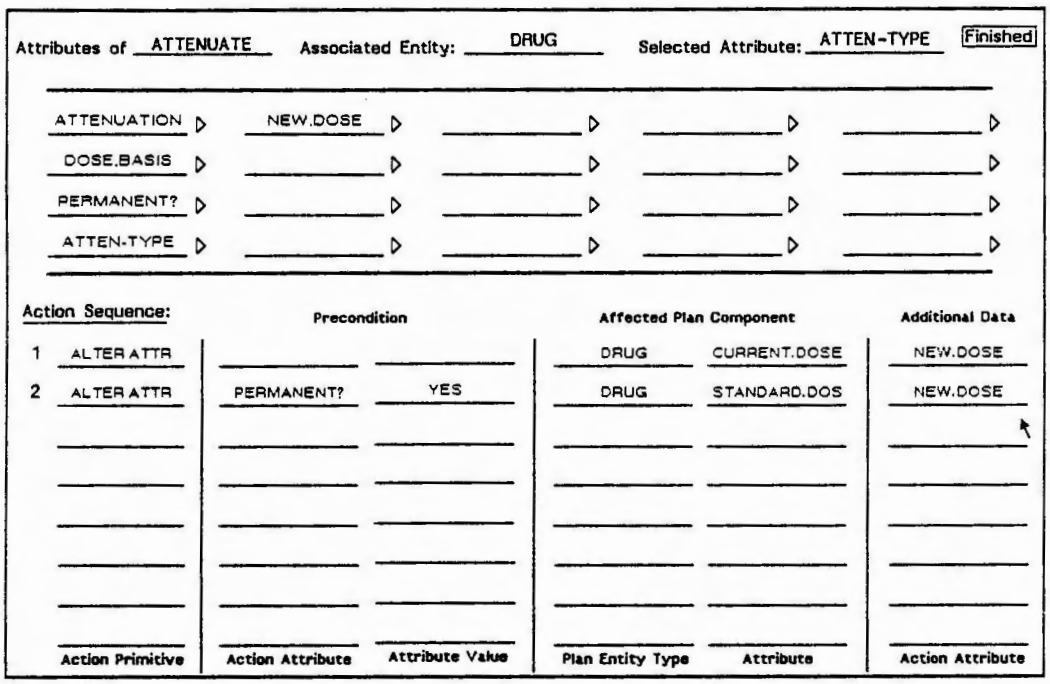


Figure 13: PROTÉGÉ form for attributes of task-level actions. The data entered into the form define the oncology *attenuate* action. There are five attributes of this action (ATTENUATION, DOSE.BASIS, PERMANENT?, ATTEN.TYPE, and NEW.DOSE). The first four of these attributes are *editor established*, and thus acquire instance-specific values from the p-OPAL user; NEW.DOSE, however, has a *computed* value that depends on the other attributes of the *attenuate* action. The two-line script in the bottom portion of the form first declares that whenever a drug is attenuated, the value of the drug's CURRENT.DOSE attribute is replaced by the value of the NEW.DOSE attribute. Then, if the p-OPAL user has indicated that the attenuation is to become permanent, the drug's STANDARD.DOSE attribute also is changed to the value of NEW.DOSE.

The descriptions of the task-level actions are used by the knowledge-acquisition tools generated by PROTÉGÉ to construct pop-up menus like those in OPAL (Figure 14). Thus, domain experts can make simple selections from menus to enter the mapping of input data values onto the corresponding actions to take.

3.3 Input Data

The third division of a task model concerns the data that the users enter during e-ONCOCIN consultations that can predicate modifications to the skeletal plan. In PROTÉGÉ, a special graphical form allows users to declare the *classes* of data that are relevant in the domain (Figure 15). Additional forms allow the user to specify the particular data *items* within each class (Figure 16). Each data class that the knowledge engineer enters into the PROTÉGÉ form in Figure 15 becomes the topic of a separate knowledge-entry form in the tool that PROTÉGÉ generates. In p-OPAL, for example, there are different forms for hematology data, chemistry data, toxic side effects, and measures of a patient's disease status.

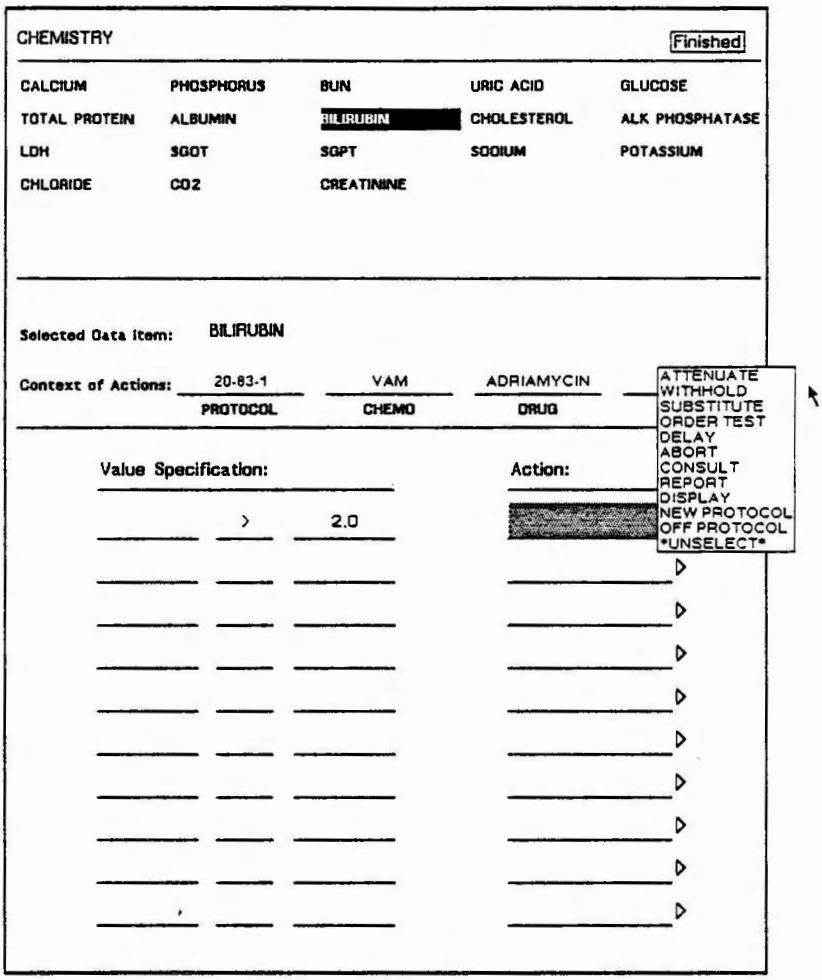


Figure 14: p-OPAL form for entering rules related to serum chemistry results. Compare this form with the OPAL form in Figure 5.

The p-OPAL form for chemistry data is shown in Figure 14. Like the original OPAL form in Figure 5, the p-OPAL form allows physicians to specify actions to take whenever certain laboratory-test results become abnormal. Although the basic structure of the form is derived from a generic template, all of the domain-specific information that the form contains comes explicitly from the task model created using PROTÉGÉ. For example, p-OPAL determines the list of chemistry tests at the top of the form from information entered at the PROTÉGÉ level.

Similarly, in the middle portion of the form, the menus that allow physicians to designate the *context* in which their specified actions apply emanate directly from the planning hierarchy that knowledge engineers entered into PROTÉGÉ (see Figure 7). As mentioned previously, the pop-up menus of task-level actions also are derived from PROTÉGÉ's explicit task model.

PROTÉGÉ creates an analogous knowledge-editor form for each class of input data in the task model. Knowledge engineers can easily change the domain-dependent appearance and behavior of such forms by editing the corresponding task model at the metalevel.

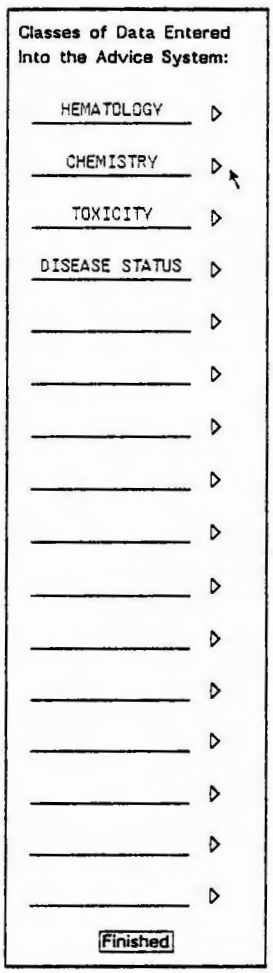


Figure 15: PROTÉGÉ form for data classes. Knowledge engineers use this form to specify the classes of input data in the task model. Selecting one of the arrows opens the PROTÉGÉ form shown in Figure 16, which allows the user to describe the individual items in the indicated data class.

4. Discussion

The methodology demonstrated by PROTÉGÉ separates the process of knowledge acquisition into a sequence of discrete steps: Knowledge engineers first use PROTÉGÉ to create and to edit task models; PROTÉGÉ then uses these task models to generate custom-tailored, graphical knowledge-acquisition tools automatically; next, domain experts use these specialized tools to instantiate the task models to define new applications; the individual tools, in turn, translate the instantiated task models into functional knowledge bases with no need for additional programming.

To date, PROTÉGÉ has been used to generate two different knowledge-acquisition tools (namely, p-OPAL and a system for the hypertension domain called HTN), both of which are capable of producing knowledge bases that can be used for e-ONCOCIN consultations (Musen, 1988b). There are many other disciplines to which PROTÉGÉ should be applicable. PROTÉGÉ will be useful for building models in any domain in which tasks can be solved via the method of

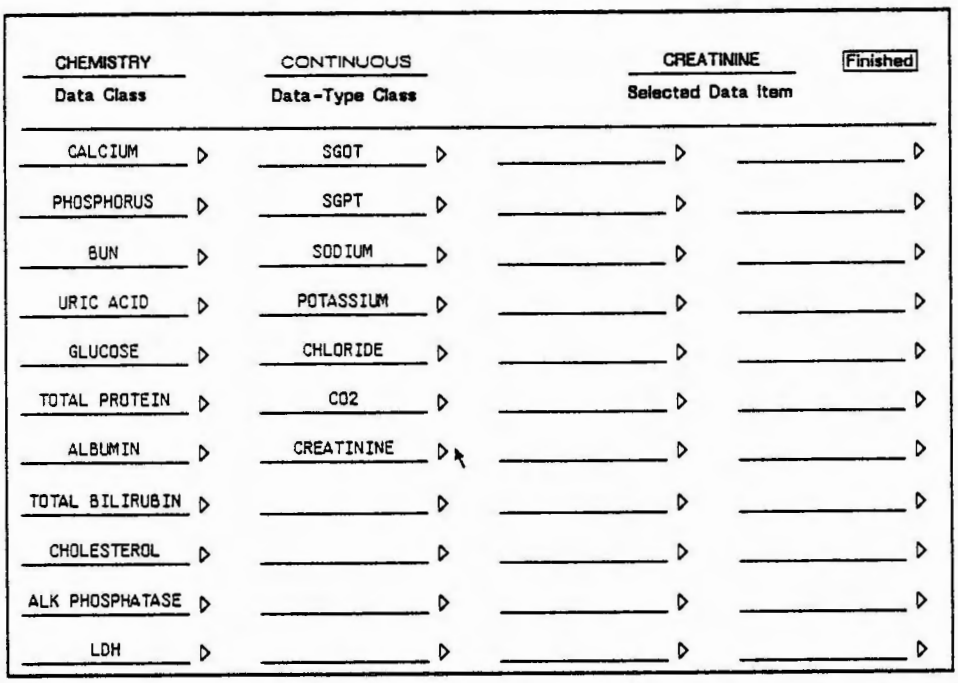


Figure 16: PROTÉGÉ form for data items. This form allows PROTÉGÉ users to list all of the anticipated data items in a data class. The entries serve to generate the menu at the top of the p-OPAL form in Figure 14.

skeletal-plan refinement (so that backtracking to satisfy constraints is not a major concern) and in which the passage of time is measured in discrete intervals—with no interval having a duration shorter than the time between e-ONCOCIN consultations. PROTÉGÉ thus should be practical for describing a number of clinical-trial areas, such as organ transplantation, diabetes control, rheumatology, and cardiovascular-drug testing.

PROTÉGÉ also could be applied to planning problems outside of clinical medicine. In Friedland's MOLGEN system (1979), for example, experiments in molecular genetics are rendered as problems in skeletal-plan refinement that have discrete steps. It would be valuable to learn to what degree *classes* of such experiments could be modeled using PROTÉGÉ, so that molecular biologists could then work with PROTÉGÉ-generated knowledge-acquisition tools to encode knowledge of specific laboratory procedures. This approach would allow advice systems based on e-ONCOCIN to aid research assistants in the execution of those experiments.

The methodology for knowledge-acquisition supported by PROTÉGÉ makes an explicit distinction between (1) creating a model to define a class of application tasks, and (2) extending that task model to define individual applications. At the PROTÉGÉ level, knowledge engineers work with domain experts to build models of tasks that can be solved using the method of skeletal-plan refinement. These models can then be used as the foundation for custom-tailored knowledge-editing tools. PROTÉGÉ is used to map out the structure of the domain and, consequently, the *process* by which an advice system might arrive at a recommendation. The tools that PROTÉGÉ generates, on the other hand, acquire knowledge about the *content* of specific protocols. The methodology consolidates at the PROTÉGÉ level the work that knowledge engineers do best—helping experts to build models; it consolidates at the level of the PROTÉGÉ-generated tools the

knowledge-entry work that experts can most readily perform on their own—specifying the content knowledge.

Where there are multiple, related tasks within an application area—and there is thus the need to construct multiple knowledge bases—this divide-and-conquer strategy offers a considerable advantage. The difficult problem of creating a computational model of the domain does not disappear, but the PROTÉGÉ approach permits system builders to confront only a single bottleneck. If knowledge engineers can help application experts to build task models by using tools such as PROTÉGÉ, those experts can then go to work on their own, defining all the individual tasks to which those models might apply. Each time knowledge engineers use PROTÉGÉ to construct a task model, application experts then can create multiple advice systems independently using the corresponding task-oriented tool that PROTÉGÉ generates.

The strategy adopted by PROTÉGÉ is founded on the notion that the major obstacle to knowledge acquisition is the task of formulating an initial model (Winograd & Flores, 1986). PROTÉGÉ accordingly provides a coherent language that simplifies the process of writing down such a model so that knowledge engineers and domain experts can readily examine and reflect on the assumptions being made. The direct-manipulation interface, extensive use of menus, and automatic bookkeeping facilities simplify the mechanics of editing an emerging task model. Yet the undeniable obstacle to knowledge acquisition—inventing the model in the first place—is, fundamentally, a human activity. PROTÉGÉ and other computer-based tools can help to stimulate the necessary creative thought, but such tools cannot determine on their own whether a complex model is sound or whether its assumptions are inherently reasonable.

Previous builders of computer-based knowledge-acquisition aids have not emphasized the distinction between the user demands of tools such as PROTÉGÉ—which help people to create and edit models—and those of tools such as the graphical editors that PROTÉGÉ generates—which help people to apply such models. Whereas the modeling of complex applications is a cognitive activity that knowledge engineers can learn to do well with experience, domain experts often lack the objectivity and the capacity to introspect that are essential for creating robust models on their own (Johnson, 1983). At the same time, considerable anecdotal evidence suggests that experts are quite effective in using a pre-existing model to define specific tasks (Musen et al., 1987; Klinker et al., 1987). The approach demonstrated by PROTÉGÉ makes a clear separation between the problem of modeling an application area and that of entering content knowledge. The division of labor should expedite the creation of expert systems for application areas in which multiple large knowledge bases are required for related domain tasks.

Acknowledgments

This work was funded by grants LM-07033 and LM-04420 from the U.S. National Library of Medicine, and grant NF-65/62-521 from the Netherlands Organization for Scientific Research (NWO). Computing facilities were provided by the SUMEX-AIM resource under NIH grant RR-00785, and through equipment gifts from Xerox Corporation and Corning Medical. Dave Combs developed much of the system software used by PROTÉGÉ. Samson Tu and Cliff Wulfman extracted e-ONCOCIN from the current ONCOCIN implementation. Ted Shortliffe, Bruce Buchanan, and Larry Fagan provided guidance for this research and participated in many valuable discussions.

References

Freiling, M.J. and Alexander, J.H. (1984) Diagrams and grammars: Tools for mass producing expert systems, in *The First Conference on Artificial Intelligence Applications,* pp. 537–543, Denver, Colorado.

Friedland, P.E. (1979) *Knowledge-Based Experiment Design in Molecular Genetics.* Ph.D. Thesis, Stanford University, Report STAN-CS-79-771.

Friedland, P.E. and Iwasaki, Y. (1985) The concept and implementation of skeletal plans. *Journal of Automated Reasoning,* **1**:161–208.

Gale, W.A. (1987) Knowledge-based knowledge acquisition for a statistical consulting system. *International Journal of Man–Machine Studies,* **26**:55–64.

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

Klinker, G., Boyd, C., Genetet, S., and McDermott, J. (1987) A KNACK for knowledge acquisition, in *Proceedings AAAI-87,* pp. 488–493, American Association for Artificial Intelligence, Seattle, Washington.

Musen, M.A. (1988a) Conceptual models of interactive knowledge-acquisition tools. *Proceedings of the European Knowledge Acquisition Workshop,* Bonn, Federal Republic of Germany.

Musen, M.A. (1988b) *Generation of Model-Based Knowledge-Acquisition Tools for Clinical-Trial Advice Systems.* Ph.D. thesis, Stanford University, Report STAN-CS-88-1194.

Musen, M.A., Fagan, L.M., Combs, D.M., and Shortliffe, E.H. (1987) Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Man–Machine Studies,* **26**:105–121.

Musen, M.A., Fagan, L.M., and Shortliffe, E.H. (1988) Graphical specification of procedural knowledge for an expert system, in Hendler, J. (ed.) *Expert Systems: The User Interface,* pp. 15–35, Norwood, New Jersey: Ablex.

Regoczei, S. and Plantinga, E.P.O. (1987) Creating the domain of discourse: ontology and inventory. *International Journal of Man–Machine Studies,* **27**:235–250.

Shortliffe, E.H., Scott, A.C., Bischoff, M.B., van Melle, W., and Jacobs, C.D. (1981) ONCOCIN: An expert system for oncology protocol management, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* pp. 876–881, Vancouver, British Columbia.

Winograd and Flores (1986) *Understanding Computers and Cognition: A New Foundation for Design.* Norwood, New Jersey: Ablex.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Acquisition of Very Large Knowledge Bases: Methodology, Tools and Applications

S. Nirenburg, I. Monarch, T. Kaufmann, I. Nirenburg

and Jaime Carbonell

Carnegie-Mellon University

Abstract.

Artificial intelligence needs to develop far more principled, manageable, and measurable approaches to the systematic construction of large-scale systems. The central component of such a scaling process must be the knowledge base. Building a very large knowledge base (VLKB) combines the scientific challenge of designing a heterogenous, interconnected knowledge representation framework that permits systematic, reliable, efficient access and update, with the technological challenge of actually constructing and demonstrating a functional, useful *knowledge acquisition and management system.* Only by addressing the problem head-on can we make rapid progress in building the infrastructure to develop multi-functional large-scale knowledge bases that are *cumulative, universally sharable, efficiently searched,* and *encoded in a semantically sound manner.*

In order to address the problem of building permanent, sharable large-scale knowledge bases, a general-purpose intelligent interactive knowledge acquisition system is needed. Its purpose is to assist knowledge acquisition teams to produce extensive domain models, dictionaries and other knowledge repositories necessary for various AI and computational linguistics applications. The knowledge acquisition and maintenance system we are developing is called ONTOS. It provides the members of a knowledge acquisition team with a state-of-the-art interactive environment which includes:

- facilities for interacting in multiple windows through menus, graphics and natural language
- memory and decision aids that help the users find the appropriate way of describing a newly-entered concept
- a filtering capability that helps identify potential inconsistencies and problems with the integrity and validity of the knowledge base.

We argue that a resident world model or ontology is indispensable for any general-purpose knowledge acquisition methodology used in developing expert systems, knowledge-based information retrieval, and natural language processing. The first large-scale application for this system is acquiring the concept lexicon for a knowledge-based machine translation project at CMU.

1. Requirements for Scaling Up AI Applications

The scientific and technological challenges of capturing and encoding human expertise in a systematic manner for inclusion in large-scale, general-purpose knowledge bases has proven far more difficult than had been envisoned. Consequently, expert systems have been highly successful only when focused on a very narrow class of problems, where the volume of knowledge required is small and manageable, and where the encoding of the knowledge can be fairly task-specific. It is time to develop systematic methods for scaling up artificial intelligence systems by several orders of magnitude. By analogy to conventional software engineering, artificial intelligence needs to develop far more principled, manageable, and measurable approaches to the systematic construction of large-scale systems. The central component of such a scaling process must be the knowledge base. Building a very large knowledge base (VLKB) combines the scientific challenge of designing a heterogenous, interconnected knowledge representation framework that permits systematic, reliable, efficient access and update, with the technological challenge of actually constructing and demonstrating a functional, useful *knowledge acquisition and management system*. Only by addressing the problem head-on can we make rapid progress in building the infrastructure to develop multi-functional large-scale knowledge bases that are *cumulative*, *universally sharable*, *efficiently searched*, and *encoded in a semantically sound manner*. This single bottleneck is perhaps the most serious problem facing AI today, both from a scientific standpoint – knowledge representation is the core of AI – and a technological one: AI must scale up its knowledge bases by several orders of magnitude in order to address real-world tasks in domains beyond the narrowly defined (though still very significant) ones of today's expert systems.

In order to address the problem of building permanent, sharable large-scale knowledge bases, many fundamental topics must be brought together in a unified project over an extended period of time:

- The knowledge representation language itself must be standardized with well-defined semantics, sufficient (or extensible) expressive power, and computationally tractable properties for update, retrieval, and multiple types of reasoning.
- an actual **VLKB** that uses the above framework and a scientifically sound semantics to store heretofore unattainable quantities of knowledge about a given set of domains
- a revolutionary **support system** for such an effort, including a state-of-the-art distributed data base transaction facility, a high-quality acquisition and maintenance environment and an effective set of on-line utilities and resources

More specifically, this large system design and implementation problem are focused on the following issues:

- How to ensure that the above scientific and large-scale technological goals are addressed **in unison.**
- **What** knowledge to represent, and at what level of abstraction, modularity, permanence and reliability.
- What **semantics** should be defined for the **representation language**, such that once encoded, the knowledge is subject to reliable and consistent intepretation.
- How to **scale up** today's AI knowledge bases by several orders of magnitude and yet retain efficient access and update akin to the much simpler but highly successful conventional data bases.
- How to build a truly **distributed** knowledge base, enabling multiple readers, writers, local caching, very reliable recovery, and efficient remote operations.

- How to provide a **broad-bandwidth interface** window into the knowledge base for browsing, question-answering, quality assurance, and similar human-oriented operations in a friendly and meaningful manner.
- How to **integrate multiple reasoning types and diverse domains** via a shared knowledge base.

Our underlying tenet is that knowledge required to perform intelligent action in real world domains is typically vast, very difficult to acquire and validate, and just as difficult to encode and transmit efficiently to consumers (whether human users or automated reasoning processes). Therefore, since knowledge is extremely valuable, both its safekeeping and its universal availability should be of paramount importance. We are developing methods of representing knowledge in a logically clean manner (thus not losing information because its encoding is semantically ambiguous or otherwise ill-determined), and one that enables efficient access by different inferential processes. A separate project at Carnegie Mellon (Camelot, see Spector, [SPB88,SS87]) is developing the underlying system architecture that will ensure both the permanence (reliability under all access and update operations in a distributed environment), and the computational tractability of the knowledge management algorithms under severe scale-up demands.

In this paper we will focus on the requirement that knowledge in a VLKB must be well-organized, its acquisition must be facilitated, and it must be stored and managed in an optimum way. However, our future work will also address other requirements: 1) A VLKB has to support multiple types of AI reasoning equally well, e.g., planning, natural language processing, diagnostic reasoning, simulation, intelligent information retrieval and machine learning; 2) Knowledge in a VLKB has to come from multiple and diverse real-world domains.

2. Acquisition and Maintenance of VLKBs

An inalienable aspect of a very large-scale knowledge base is its underlying computational environment. Such an environment must include fast access and retrieval mechanisms which tie together the knowledge representation and its attendant distributed data base system. These facets of the task of building VLKBs offer somewhat different perspectives on knowledge representation. We will describe them in the next section, after which we will discuss the appropriateness and feasibility of automatic acquisition as opposed to manual. Descriptions of the desiderata for a knowledge acquisition and maintenance system and of ONTOS, our knowledge acquisition tool, complete this chapter.

2.1. Views at Knowledge Representation

The way knowledge is represented can, of course, be determined by the nature of a particular application, such as information retrieval or expert system design. At the same time, the various basic functionalities for handling knowledge also introduce desiderata with respect to knowledge encoding and maintenance. A knowledge representation formalism as such has a preference for simplicity of syntax, facilitating access through efficient indexing and search, conceptual economy of storage through an inheritance mechanism, etc. A general world model requires meaningful constraints on the choice of categories and properties, on the sets of properties that define concepts, and on the legal sets of values for particular relations and attributes. A data transaction facility requires specific knowledge about actual updates and accesses to a frame, in general, about data dependencies and about atomic transactions. In what follows we attempt to describe the three perspectives on knowledge mentioned above, as they pertain to our knowledge acquisition and maintenance system.

2.1.1. Knowledge Representation Language Prespective: FrameKit

The underlying knowledge representation framework in our system is FrameKit, a frame-based knowledge representation language (Carbonell and Joseph, 1985; Nyberg, 1988). The syntactic objects FrameKit recognizes are frames, slots, facets, views and fillers. Each frame can have any number of slots. Each slot can have any number of facets, each facet can have any number of views, and each view can have any number of fillers. FrameKit also provides built-in facilities for manipulating data. Frames refer to named sets of property values. Slots name individual properties. Facets allow listing several types of information about a slot, in addition to its value. The fillers within a single facet can be partitioned into subsets corresponding to different views. A filler can be any Lisp object.

The above syntactic objects have certain intended semantic interpretations. Some examples of these intended interpretations are as follows. A subset of frames are *relations*, that are used to represent (named) links between frames. With the help of these relations one can, for instance, organize sets of frames into taxonomic hierarchies. A data manipulation mechanism called *inheritance* is defined in FrameKit to alleviate redundant storage. A special semantic class of fillers are *demons*, which are procedural attachments that can be used in various knowledge-intensive applications, typically, to support control decisions in blackboard and other data-directed architectures. *Views* are used to constrain the scope of certain data that should only be accessible to certain processes.

This small scale and basic functionality, when contrasted with more complex knowledge representation systems, translates into greater computational efficiency and ease of use.

2.1.2. Data Transaction Perspective: Camelot

It is clear that there can be uniform systems support for the synchronization, reliability, security, and distribution aspects of AI knowledge bases. Such systems support is valuable because it reduces the cost of developing AI knowledge bases, and allows the combined use of AI and traditional data bases. However, there must be great flexibility in developing or enhancing data models that permit efficient storage and retrieval of large quantities of knowledge.

The semantics of our frame-based knowledge representation language is defined only in terms of the abstract data types. These data types are supported by underlying data models based on networks of frames. For the purposes of supporting data transactions, the data type *frame* can be defined to consist of:

- A header (unique internal name, type declaration, and creation/update record)
- The source of the knowledge in the frame (external, internal, fallible, unimpeachable, etc.)
- A set of tuples <*slot, value, constraints, view, dependencies, augmentation*>, where:
  - A *slot* is a local attribute name, and *frame.slot* pairs are globally unique names.
  - A *value* is a set of atomic elements, locally defined frames, or names of external frames.
  - *Constraints* are predicates whose conjunction must be satisfied by any candidate value.
  - A *view* is a structured object specifying who has access to the value of the slot and how the information should be transmitted to the requesting process.
  - *Dependencies* list other parts of memory (frames or slot values) from which the local value was calculated or propagated (and therefore depends upon), as well as inverse dependency links.

- *Augmentations* are user-definable structures to encode systematic inference, such as forward propagation of the value when it is updated ('if-added'), backward propagation to seek the value when a read request fails to find a local value ('if-needed'), or to invoke calculations external to the frame system, such as computation in a certainty calculus (or classical probability) of the degree of belief in the value, given the current state of memory (including dependencies).

- The *inheritance topology* including an abstraction hierarchy (*is-a* links and their transitive closure graph), a partonomy (*part-of* links and their closure), plus other domain dependent inheritance graphs (spatial or temporal adjacency, causality, lines of authority in human organizations, lines of data flow in computer systems, and so forth).

2.1.3. Ontological Modeling Perspective

To facilitate the construction of ontological domain models, we need to extend the semantic definitions and constraints already available in a knowledge representation language such as FrameKit. Many of these extensions refer to the semantics of slot (or,in FrameKit, facet) fillers.

In this section we will discuss the semantics of the FrameKit knowledge representation used in ONTOS, a knowledge acquisition and maintenance system developed at the Center for Machine Translation at Carnegie Mellon University[1]. Each frame represents an ONTOS concept. Concepts are subdivided into the types of things that can be refered to, such as objects and events, and properties. Properties of concepts in ONTOS are subdivided into relations and attributes. Relations map concepts into concepts, while attributes map concepts into value sets. Thus, for instance, *part-of* is a relation – consider (*part-of* wheel car); while *age* is an attribute, whose value set is defined in ONTOS as a range from 0 to 1,000,000,000,000 seconds[2]. Another example of an attribute is *color*, which in ONTOS maps physical objects into a set of symbols (a value set) without a strict ordering[3].

The distinction in ONTOS between relations and attributes captures the intuitive ontological distinction between them, namely that relations involve two or more things in the world, whereas attributes only involve one. Making reference to an element in a value set is not referencing one thing in terms of another, but referring to a given thing in a certain way. This method of encoding an ontological distinction has computational benefits as well. It establishes a syntactic criterion that facilitates consistency and type checking in creating and extending a knowledge base.

There are a number of legal types of facet fillers in ONTOS. First of all, the fillers can be either given directly or referenced through the value of another facet or slot, possibly in a different concept (note that the underlying knowledge representation language guarantees the uniqueness of all (frame slot) pairs). ONTOS employs the 'dot' notation – <frame-name>.<slot-name> – as its reference mechanism in these cases.

There are the following types of the directly listed values:

---

[1]This section describes only the ontological constraints used in building domain models using ONTOS. The architecture of the system and its functionalities are described in in Section 2.4 below

[2]All scalar attributes are defined using standard units of measurement, seconds in the case of temporal ranges. The boundaries of ranges in ONTOS are subject to change depending on domain and application requirements.

[3]We have considered using a quantitative standard for color, such as the Munsell color notation, which specifies colors on numerical scales of hue, value, and chroma and can be expressed as accurately as desired. The boundaries of each color name can be fixed and defined in terms of this or some like numerical notation.

1. Single concept names
2. Multiple concept names
3. Single value set elements
4. Multiple symbolic value set elements
5. Multiple numerical value set elements

Our ongoing concern is the extension and modification of the currently available model of the world to an appropriate level of detail, while at all times safeguarding its theoretical and empirical soundness. Such a model, which we call an *ontological* model, provides uniform definitions of basic semantic categories (such as objects, event-types, relations, properties, episodes, and many more) that are the building blocks for descriptions of particular domains. This world model is organized as a multiply interconnected network of frames. A knowledge representation language with all the properties discussed above serves as a grammar of the *lingua mentalis* which we use for describing the world. The center of our attention is, however, the task of defining and creating the **lexicon** of this *lingua mentalis*.

2.1.3.1. Semantic Memory The requirements of knowledge representation extend above and beyond providing an adequate formalism in which to record knowledge, and adequate means of storage and retrieval. One has to specify the *contents*, the semantics of the knowledge units. This task is less formalizable than the syntactic aspects of knowledge representation, and this is one reason why relatively little progress has been made in AI with respect to ontological world modelling. Even today this area of scientific research remains, as it has been for over 2,500 years, within the purview of philosophy. While a number of important theories have been propounded in philosophical ontology, we believe that it is necessary to reformulate the goals and methodology of this inquiry, similarly to the way the finite time/space constraints of practical computation changed the style and attitudes of certain areas of discrete mathematics, which gave birth to the theory of computation.

An ontological model must define a large set of generally applicable categories for world description. Among the types of such categories are:

- Perceptual and common sense categories necessary for an intelligent agent (such as, for instance, a planning or learning program) to interact with and manipulate the states of the outside world (see articles in [Hob85] and [For83]).
- Categories for encoding interagent knowledge which involves 1) one's own as well as other agents' intentions, plans, actions and beliefs and 2) communication-related knowledge, including syntax and semantics of at least one natural language as well as discourse and pragmatics rules of linguistic communication [Nir85].
- Categories that help describe metaknowledge (i.e., knowledge about knowledge and its manipulation, including rules of behavior and heuristics for constraining search spaces in various domains).
- Means of encoding categories generated through the application of the above inference knowledge to the contents of an agent's world model (see articles in [BL85]).

The choice of categories is not a straightforward task, as anyone who has tried realistic-scale world description knows all too well. Here are some examples of the issues encountered in such an undertaking:

Which of the set of attributes pertinent to a certain concept should be singled out as *concept-forming* and thus have named nodes in the conceptual network corresponding to them, and which other ones should be accessible only through the concept of which they are properties? As an example consider whether you would further subdivide the class *vehicle* into *water-vehicle*, *land-vehicle*, *air-vehicle* or rather into *engine-vehicle*, *animal-propelled-vehicle*, *gravity-propelled-vehicle*; or maybe into *cargo-vehicle*, *passenger-vehicle*, *toy-vehicle*, *mixed-cargo-and-passenger-vehicle*? Or maybe it is preferable to have a large number of small classes, such as, for instance, *water-passenger-animal-propelled-vehicle*, of which, for instance, a rowboat will be a member?

Which entities should be considered *objects* and which ones, *relations*? Should we interpret a cable connecting a computer and a terminal as a *relation*? Or should we rather define it as a *physical-object* and then specify its typical *role* in the static episode or 'scene' involving the above three objects? Should one differentiate between *relations* and *attributes*? Or define attributes as one-place relations? Is it a good idea to introduce the ontological category of *attribute value set* with its members being primitive unstructured meanings (such as the various scalars and other, unordered, sets of properties)? Or is it better to define them as full-fledged ontological concepts, even though a vast majority of relations defined in the ontology would not be applicable to them (such a list will include case relations, partonomy, ownership, causals, etc.)?

Should we represent colors *symbolically*, as, say *red*, *blue*, etc. or should we rather define them through their spectrum wavelengths, position on the white/black scale and brightness? How should we treat *sets* of values? Should we represent *The three musketeers* as one concept or a set of three? What about *The Pittsburgh Steelers*? What's an acceptable way of representing *complex causal chains*? How does one represent a concept like *fake gun*? Is it a gun? Or a toy? Or none of the above? Or is it maybe the influence of natural language and a peculiar choice of meaning realization on the part of the speaker that poses this problem – maybe we don't *need* to represent this concept at all?!

Whereas we advocate scaled representations discussed above, we must, for any level of detail, provide concrete answers to these questions. A number of such decisions have already been made in our work on the prototype ontological model which we acquired using our prototype knowledge acquisition and maintenance system ONTOS. Figure 1 shows several subnetworks from this ontology produced by ONTOS from its resident ontology with the help of its browsing facility. This output displays some of the answers we suggested for questions of the type sampled above.

2.1.3.2. Episodic Memory A general-purpose world model must cover the union of needs for multiple application domains and for any type of rational reasoning. In many domains it becomes important to support reasoning from the point of view of modelling multiple intelligent agents. The knowledge that such agents possess and actively use to solve problems in various knowledge domains and types of activity includes not only an ontological world model, as described above, but also records of past experiences (including learned or reported ones). To continue with the linguistic analogy, in addition to grammar and lexicon our *lingua mentalis* makes itself manifest in 'texts' encoded in it.

The *lingua mentalis* equivalent of a text is an *episode*, a unit of knowledge that encapsulates a particular experience of an intelligent agent, and which is typically represented as a temporally and causally ordered subnetwork of frames.

The ontology and the episodes are sometimes discussed in terms of the contents of two different types of memory: semantic and episodic [Tul85]. This distinction seems useful in computational modelling as well. In our work we are representing, with a varying degree of specificity, both ontological concepts and episodes. The same knowledge representation apparatus is used for storing the contents of both the ontology and the episode cluster.

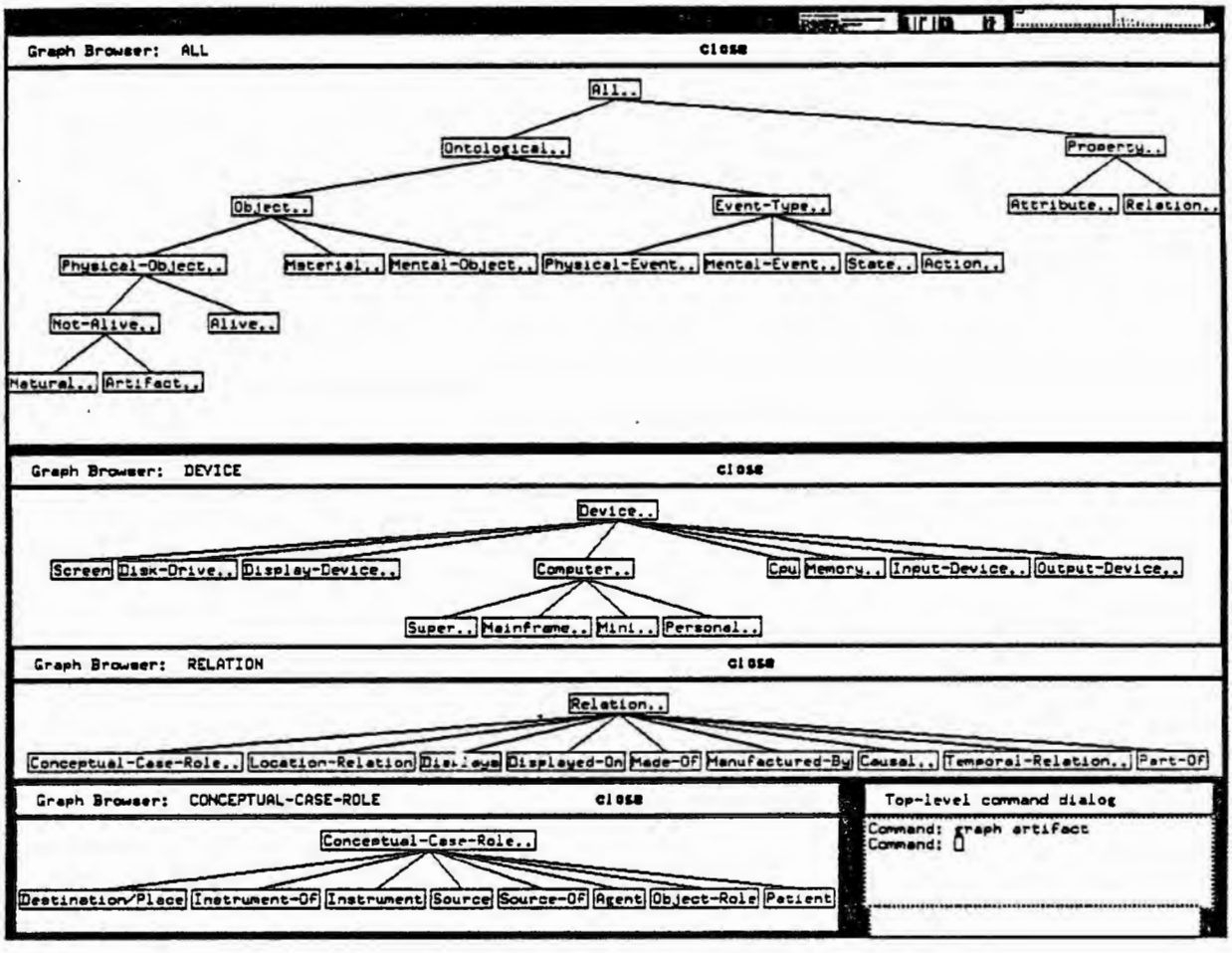


Figure 1: A subset of a currently available ontology; graphical view on an ONTOS screen.

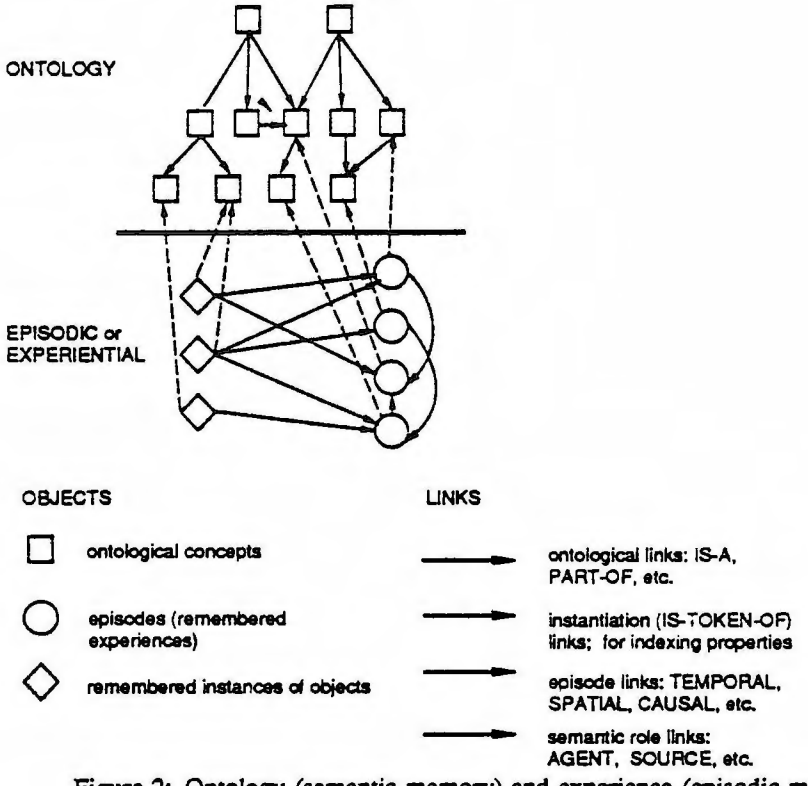


Figure 2: Ontology (semantic memory) and experience (episodic memory).

It becomes important, from the theoretical, descriptive and implementational points of view, to define an adequate interface between the two memories. We suggest the following approach. The episodes (remembered experiences) are temporally and hierarchically organized sets of tokens of event-types. They are indexed through the type they correspond to and can be interrelated on temporal, causal and other such links. The participant roles in the episodes can be either instantiations of object and event types in the semantic memory or references to existing named instances, stored outside semantic memory, but having links to their corresponding types (see Figure 2). The presence of a systematic representation and indexing method for episodic knowledge is an enablement condition for case-based reasoning [Kol84,Sch82] and analogical inference [Car83,Car86].

2.2. Interactive or Automatic?

Describing a large domain is a complex and labor-intensive task. Since it is not possible at present to perform it automatically,[4] one must use the available AI, NLP and generally computer technology to design and build aids for this type of knowledge acquisition. We call such aids **knowledge acquisition and maintenance systems**, KAMS.

An integral part of the VLKB project is support for various methods for acquiring new knowledge, for focused retrieval, for less-focused navigation and browsing through the structured knowledge base,

[4] One would presume, or perhaps, hope, that some human labor would always be necessary.

and for using the knowledge itself to help guide these processes. The vast majority of organized human knowledge exists in textual form, and human communication always resorts to natural language, regardless of the technological revolution. Therefore, we consider extraction of knowledge from texts to be of utmost importance. Whereas we presently conduct manual entry of knowledge, we wish to prepare for automation in the future. Equally important is the ability to render structured knowledge in frame-based representations into natural language on output for human comprehensibility (whether for simple question-answering, or more elaborate computer-based instruction).

2.3. The Architecture of a KAMS

At present, one's best bet for acquiring a VLKB is manually, with the help of an intelligent interactive aid (cf. Lenat and Feigenbaum, 1987). A KAMS should include the following basic components: a) a *data transaction* component; b) a *user interaction* component; c) a *communication* component; and d) a *resource and utilities* component.

2.3.1. The Data Transaction Component

The *data transaction* component of a KAMS must provide at least the following functionalities: a) efficient *storage* organization and *search* techniques for the resident knowledge; b) a state-of-the-art *data manipulation language*; and c) a variety of *consistency* and *validity* checks on the information introduced by knowledge enterers into the resident knowledge.

The actual implementation of frames and the inheritance graphs among them lies beyond the scope of this document and the purview of ONTOS. See [SPB88,SS87] for descriptions of the Camelot data transaction facility which supports this functionality in ONTOS.

2.3.2. The User Interaction Component

The *user interaction* component of a KAMS should include at least the following capabilities: a) a structured screen *editor* customized for the knowledge acquisition and maintenance requirements of a particular application; b) a graphics-based *browser*; c) an *information retrieval* subsystem; d) a *natural language interface* facilitating information retrieval from the resident knowledge body as well as from ancillary knowledge sources; e) a means of *interaction* among the above functionalities (modes of operation); and f) a flexible and extensive *on-line help* and 'suggestions' facility.

2.3.3. The Communication Component

The *communication* component supports effective communication among the members of the knowledge acquisition team and the team manager. Such a component is necessary to minimize judgment inconsistencies and to maintain the integrity of the knowledge base.

2.3.4. The Resources and Utilities Components

The *resource* component of a KAMS helps the knowledge enterer to make appropriate description decisions. It contains various reference sources: *resident knowledge*, the knowledge already in the knowledge base; relevant (machine-readable) *dictionaries* and *encyclopedias*; and extensive *text corpora* in the appropriate application domain.

The *utilities* component includes at least the following: a spelling checker, a text-oriented statistical package (including, for instance, a concordance-building capability), a morphological processor and a syntactic analyzer for every NL used in the interfaces.

In the next section we illustrate the above actual capabilities of our KAMS, ONTOS.

2.4. ONTOS Functionalities

We implemented the *interaction* and the present *DBMS* components of the KAMS in CommonLisp, using the X window system under the Mach operating system on an IBM PC RT. The resident knowledge (the contents of the current *resource* component) can be expressed either in the FrameKit [CJ85] or PARMENIDES [She87] frame-based knowledge representation languages. This prototype KAMS is called ONTOS.

The current functionalities in the interaction component include a customized structured editor that facilitates editing of frames, a graphics browser (that supports the visual representation of the various networks, the direct-manipulation (mouse-driven) display and editing functionalities), a menu-driven interface to top-level commands (manipulating frames and saving and printing capabilities), and a first version of the on-line help and suggestions facility.

The current functionalities of the DBMS component include checking whether or not a newly created node is subsumable under or identical to its parents and/or siblings, and checking the frame and slot name appropriateness as well as the types of slot values.

Figure 3 and 4 illustrate the current knowledge entering capabilities of ONTOS. At the start of a dialog for entering a new concept, ONTOS requests its **is-a** value. In figure 3, we are assuming that the knowledge enterer has already (see figure 1) browsed through the **is-a** hierarchy of **physical object** and is now looking at the subclass of **personal** computer and that he has already determined that the concept of **macintosh** belongs under **personal** computer. When the **is-a** value for **Macintosh** is entered, a window is opened with the initial information inherited from **personal** computer on up. Since **IBM-PC** is the only child of **personal** computer, it probably contains other information of use to the enterer, so he has ONTOS show the frame specification for this concept as well. Figure 4 shows how the inherited information and sibling information provide a template for the user to specify the concept **Macintosh**. Notice also that a new property has to be defined, **Macintosh-make** (see figure 4), since **IBM-makes** and **Macintosh-makes** have to be distinguished.

2.5. ONTOS Applications: the Example of NLP

In order to develop a successful NLP application, a detailed model of the domain served by a sublanguage must be constructed **prior** to any linguistic work, as in Hayes' method of domain-driven definition for entity-oriented parsing (see [CH88]). The domains used for NLP applications are usually better defined

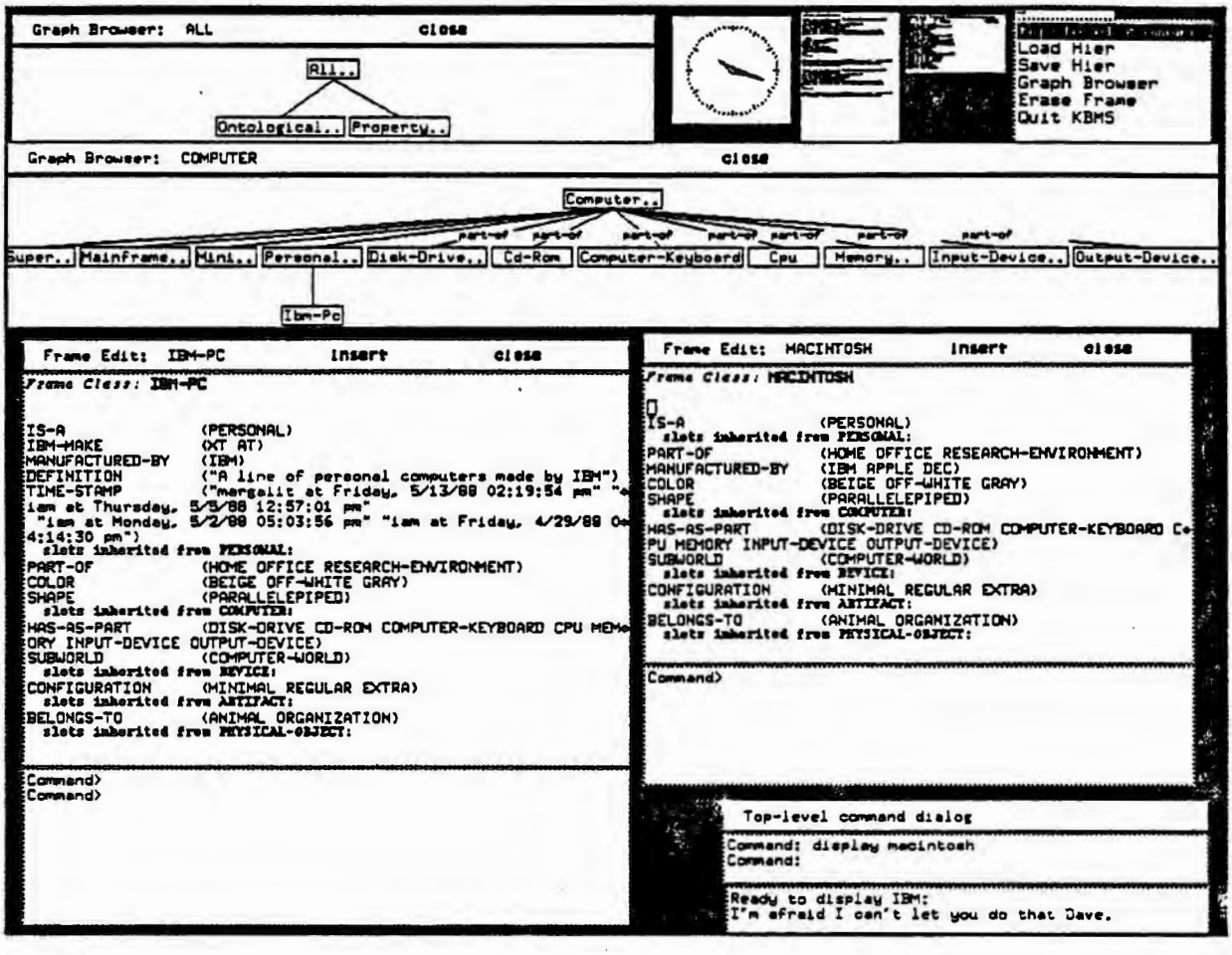


Figure 3: The Concept of a Macintosh Computer in its Initial Form

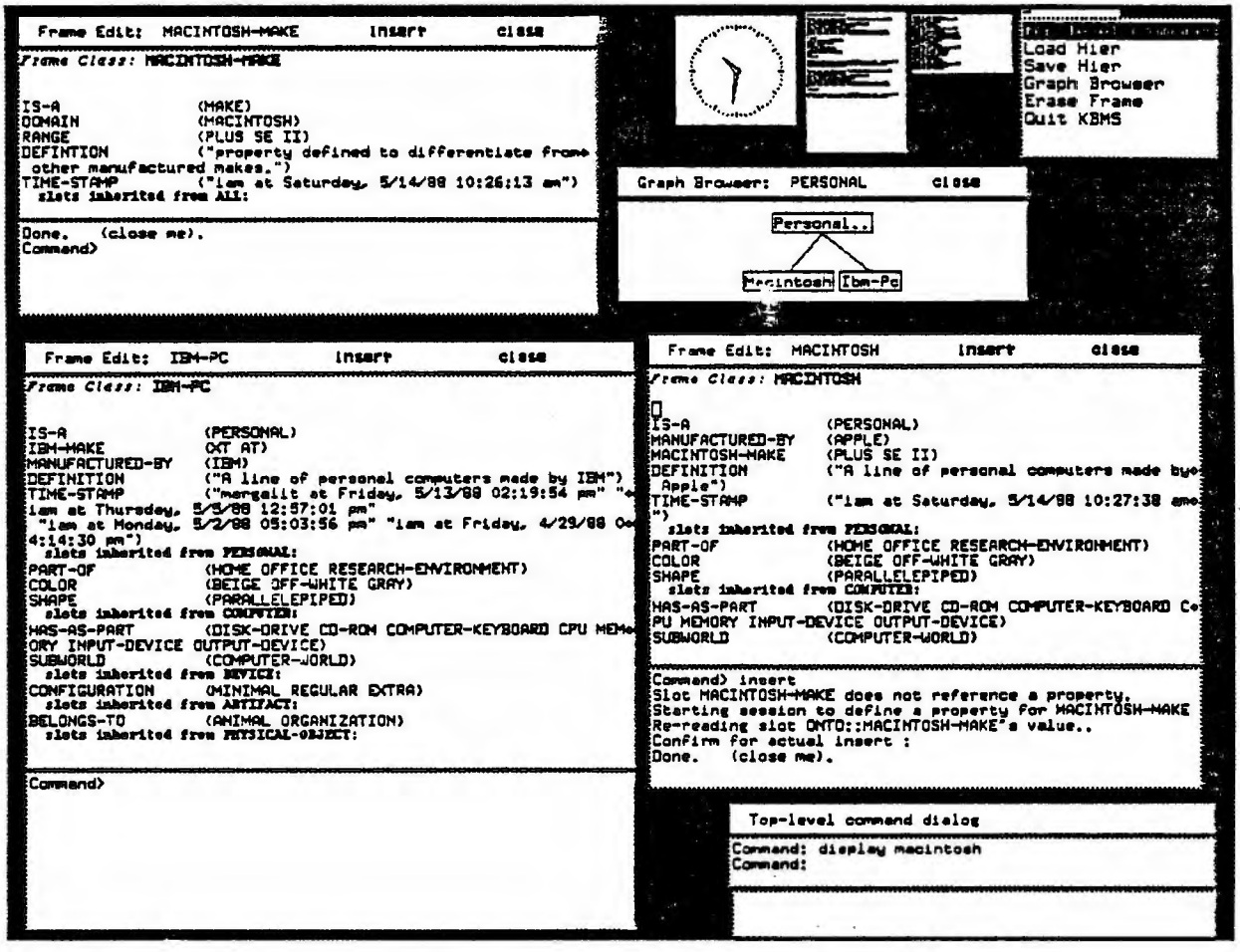


Figure 4: Adding Properties on the Basis of Inheritance and Sibling Differentiation

and structured than other areas of the world. They are usually organized around a reasonably clear and user-uniform conceptual taxonomical *is-a* hierarchy, and the sublanguages serving them are to a large extent terminological or semi-terminological. Even more importantly, the grain size (or granularity, [Hob5a]) of the required meaning processing in NLP systems is usually considerably larger than in a natural language as a whole [NR86], p. 338.

For the purposes of the NLP type of reasoning we agree with the Tacitus [Hob86], p.220, methodological position 'to define rich core theories of various domains... and then to define... English words in terms of predicates provided by these core theories.' We believe, however, that our notion of ontology presupposes a richer world description than the level aimed at in Tacitus. Thus, our view of the relationship between VLKB and NLP reasoning is made manifest by our firm commitment to acquiring an ontological domain model prior to describing the lexis of particular language. Indeed, in a computer application one needs world model units as elements of the metalanguage for describing the meanings of words and phrases in a natural language.

This program of knowledge acquisition for NLP is being implemented in one of the machine translation projects at the Center for Machine Translation. The recognized necessity to describe the concept lexicon prior to dealing with the lexis of natural languages is a feature which clearly distinguishes our approach from most other work on computational lexicography and brings us much closer to non-NL-oriented work on knowledge acquisition.

The above lexicon acquisition paradigm is described in much greater detail in [NC87] [NR87a], [NR87b], [Nir88]. Our experience with using ONTOS to acquire the domain knowledge for the domain of IBM PC installation and maintenance has been extremely successful, since it significantly speeded up the process of domain knowledge specification. At present we are implementing the interaction dialog for the acquisition of English and Japanese lexicons that describe the meanings of words and phrases in terms of the resident ontology. The integration of this capability into the set of ONTOS functionalities is made easy by the 'open-system' type of design we use in ONTOS. After the dialogs are completed, it will essentially consist in adding an additional field in the top-level ONTOS menu.

3. Further Applications of ONTOS

Viewed as an advanced tool for knowledge acquisition and maintenance, ONTOS can find use in any knowledge-intensive task where rapid prototyping and ease of debugging are crucial — which, essentially, means any large-scale AI application. It is clear that tools of this class will become the standard in the knowledge industry tomorrow, supplanting non-ontologically based interfaces, such as KEE or KnowledgeCraft, which are the standard today.

The fact that ONTOS facilitates the acquisition and maintenance of large domain models makes it especially attuned to certain types of applications. One of the ONTOS applications — natural language processing — has been briefly described above. Another promising application of ONTOS could be in a testbed for multiple-agent planning and problem-solving systems. In such an application it is essential to maintain a 'belief space' for each of the agents and reason about their similarities and differences. Any realistic scale experimentation would require fast and convenient methods of updating the agents' world models. And ONTOS can provide that capability.

In what follows we will describe in some more detail how ONTOS facilitates the design and implementation of yet another important knowledge-oriented application.

3.1. Extensions to ONTOS: Conceptual Indexing

The amount of publicly accessible, relatively unstructured or inadequately structured, on-line textual information (on-line text corpora, machine readable dictionaries and encyclopedia etc...) is already quite extensive and will, no doubt, grow significantly in the near future. This is of special significance to scholars and researchers in the sciences and technology. Schemes need to be developed to aid database maintainers in organizing and indexing huge amounts of textual information[5] as well as integrating the results with other forms of on-line information, such as numerical data, graphs, diagrams, and pictures. These schemes must enable a variety of users to comprehend, quickly and effectively, the global import of such integrated information and then to focus on those elements that are specifically new and relevant. We believe that representing the structure and organization of human knowledge is the key for structuring, indexing and searching such on-line textual information. This is why ONTOS is being extended to support knowledge-based indexing of on-line texts and their transformation into interconnected multi-dimensional information structures.

VLKBs necessary for natural language processing and reasoning in real-world applications are also necessary for structuring of electronic text. Cross-referencing and linking texts and text fragments requires that relevant concepts in large numbers be represented at the appropriate level of granularity. The ontology resident in ONTOS and the domain models of shared knowledge it facilitates the creation of conceptually indexed networks of texts, as well as provide a means for subsequent navigation through them. The conceptual network that is built by using ONTOS and its resulting graphic presentation can also be used to generate graphic representations of the propositional content and discourse connectivity of text, as well as content-based links among separated portions of text. Different portions of text will have different graphic networks representing them. These graphic networks also provide a basis for formulating queries in graphic form and matching them to portions of text represented by similar graphic forms. The meaning of a concept can be edited using the facilities of ONTOS according to the representational requirements of an indexer or information seeker by selecting and/or excluding slots and constraining slot fillers. The ranges expressed by slot fillers can be modified by recursively editing the meanings of concepts or restricting the property values in these ranges. Since the conceptual network built with the use ONTOS connects every concept node to every other concept node, users will be able to build conceptual graphs of queries and texts.

As we described above, we are presently using ONTOS to aid in building a conceptual network that serves as the basis for the semantic module of a knowledge-based machine translation system. The same conceptual network that is being generated in ONTOS for the translation system also serves as a hyper-index for the very on-line text being translated. The text is a computer installation and maintenance manual.

Two modules are being added to ONTOS in order for it to support these indexing and information retrieval facilities:

- An information retrieval system that supports query building and retrieval through conceptual indexing.[6]
- A dictionary which provides grammatical information about words and phrases and pointers from these into the conceptual-frame network.

Besides a definition slot many ONTOS frames will contain at least one other text slot. For example:

---

[5] A precursor to our work in conceptual indexing is described in [Wal87].

[6] Conceptual indexing, as opposed to keyword indexing, provides the means to represent relations. With these means the difference between, for example, the 'economy of science' and the 'science of economy' is representable.

textual-instance: is a slot containing pointers to textual instances of the concept specified in the frame. It will be the basic way of linking the ONTOS conceptual network to a text database or hypertext.

The textual instances are spans of text such as a word, a phrase, a clause, a sentence, a sequence of sentences, or even a sequence of paragraphs. Different spans of text overlap and are indexed into different concepts. Texts and portions of text are selected or retrieved and viewed in separate windows. Texts adjoining those selected or retrieved can also be included in these windows. These adjoining texts can be highlighted at word, phrase, clause or more encompassing levels at the request of the user. These highlighted units will be connected, in turn, to concepts in the conceptual network. Graphic representations of these will also be displayed in separate windows, according to some selectable depth and breadth. In an on-line concept- or knowledge-based hypertext system, the index is no longer a tangential part of a text, removed from sight. Rather, the index, conceptually-based and graphically presented, is always in view, an intrinsic part of the way the text is read and used.

We are in the process of automating procedures for building conceptual indexes and information retrieval systems (see also next section). This involves a number of steps, some of which are already on-going or about to be under way and some of which are more distant. These include:

1. Building a domain specific dictionary which provides grammatical information about words (including nouns, verbs, adjectives and adverbs) and phrases with pointers back and forth from these to the conceptual-frame network. This is tantamount to adding a list of synonyms and more to each node in a conceptual network. Each synonym has the meaning specified by the frame at that node with respect to a certain degree of granularity.

2. Using free text database tools to create inverted word and phrase indexes of domain specific texts, filtering common words and morphological variation. Phrases that have a unified conceptual meaning may be detected through a relatively high frequency of occurrence.

3. Implementing a system that will automatically create a pointer for each word or phrase in the inverted text index[7] to the concept in the ONTOS built domain representation, as well as a pointer to the word or phrase in the dictionary described above. Not all words contained in phrases will have corresponding conceptual pointers.

4. Writing procedures that will enable:

   (a) user selected concepts defined with the use of ONTOS to be specified as a conceptual graph of a query or text.

   (b) conceptual graphs representing texts to be compared with conceptual graphs representing queries.

   (c) selection of texts or portions of text for viewing which most closely match (thresholds should be user specifiable) the queries.

For words or phrases not appearing in the dictionary information associated with a concept, non-automated procedures will still be necessary. Basically this involves using the facilities already available in ONTOS to find an appropriate cluster of concepts by browsing graphs and frames. This may involve updating the knowledge base to cover conceptual inadequacies in the domain representation. Whether or not new concepts are necessary, words and phrases not appearing in the dictionary have to be entered into it and associated with the appropriate concepts.

---

[7] weighting the word or phrase primarily thru user selection

3.2. Extensions to ONTOS: Incremental Automation

Many workers in knowledge representation believe that once a notation for knowledge representation is developed, the task of using it actually to describe the world may prove to be relatively simple. This attitude seems counter-intuitive and counter-experiential, as any developer of an NLP system and/or field linguist can attest. In any large application, lexicon building has to be done by a large number of relatively untrained people who are certain to make non-uniform decisions in many cases. The instructions these people obtain cannot be very precise, unambiguous or designed to provide in advance for each contingency. And these are some of the reasons why even a good knowledge representation language alone cannot guarantee a quality lexicon.

Fully automated lexicon building procedures would definitely ensure the uniformity of description. There was a considerable enthusiasm about the possibility of a complete automation of lexicography some 15-20 years ago (see, for instance, [Col60,Gri70,Kuc69,Ven73]. A more *feasible* scenario, however, is partial automation, for example, along the lines suggested in this paper and elsewhere [NR87b].

The NLP researchers' interest in lexicons has increased dramatically in the 1980s not only because of the increased interest in NLP semantics but – more importantly – because of the emergence of **machine-readable dictionaries** and the growing realization of their role as an important resource in NLP. Recent work on machine-readable dictionaries offers new and interesting possibilities both for computer-assisted lexicography and for the construction of lexical databases derived from the definitions in machine-readable dictionaries and utilized in NLP along with other fields (see [Wal84,Cal84a,Cal84b,Ahl85,FWS*86]).

The availability of machine-readable dictionaries has led to interesting work in the area of knowledge representation and to methods for the automated extraction of various forms of semantic information, such as relations and hierarchies [BBC*87,CB85,WFG*87,CG87].

Most of the knowledge representation work in NLP has been done in connection with building semantic analyzers. The two major approaches to this task have been inspired by the work of Yorick Wilks [Fas86,WF84,WFG*87] and Roger Schank [CCG81,BS80,Cul86]. The latter school made a number of additional important contributions to the area of knowledge representation [SA77,Car81,Wil83,Sch82].

We have been experimenting with some automated methods. Much of human knowledge is already stored in computers as natural language text. One desirable method for building VLKBs would be for computers to use natural language understanding to read dictionaries, encyclopedias, manuals, journal articles and other texts for automatic extraction and reformulation of the underlying knowledge structures in these texts. Several projects are already working on this kind of machine learning. The RESEARCHER system [Leb85] reads patent applications to generate computer models of the machines described by the patent. The TESS system [YH85] reads bank telexes to extract information about the financial transactions they represent. The FRUMP system [DeJ79] could read, understand and translate stories from the UPI newswire. The FERRET system [MCT87] can read texts from the Dow Jones newswire and the UseNet international newsgroup system. What all of these systems have in common is that they read real texts and extract useful information from them, representing their output as interrelated networks of frames.

Two of these systems, RESEARCHER and FERRET incorporate another kind of learning: they are self-extending. As they process more and more texts, they get better at processing the material they are designed to understand. The RESEARCHER system learns by generalizing the descriptions of the machines it reads about. The FERRET system uses a trial-and-error method called genetic learning [Hol75] to acquire new structures to represent the new kinds of concepts and relations encountered while reading.

4. Survey of Research and Development of Knowledge Acquisition Systems

Knowledge representation work in AI has paid relatively little attention to its ontological foundations. We believe that an adequate knowledge representation theory must include a resident ontology which supports knowledge acquisition. In what follows we will survey the work in this area and compare our approach to those of others.

4.1. World Models

Whereas knowledge representation has been a central concern in artificial intelligence, very few investigators have been concerned with the interlocking of theoretical and practical problems involved in building VLKBs. Some important early articles are included in the collected volume *Associative Networks* [Fin79], where Brachman [BL85] summarizes the state of the art in semantic networks and frame-based systems. Some of the earlier systems, such as the various conceptual dependency knowledge representation languages, were of particular importance to natural language processing, e.g., [Sch75,Wil84,Cul86,Kol84,Jac85]. Thus far, most knowledge bases have focused on exploring specific paradigms such as KRL [BW77], and KL-ONE/KL-TWO/KRYPTON family [BS85,BFL83], but have kept their knowledge bases relatively small.

Some have argued that a carefully interpreted frame representation language supporting automatic subsumption of structured types need not concern itself with any ontological scruples [LB85] or build in knowledge of natural kinds, e.g., NIKL [KR86]. But knowledge acquisition is more than just automatic subsumption. We believe that for automatic subsumption to procede properly, ontological categories, including what are called natural kinds, have to be specified first. This includes basic categories for slots or roles, that is, basic properties and relations. Our strategy is to provide a lot of top level support, so that adding new concepts becomes a focused and simple editing task through the use of inheritance. Once a large portion of ontology and a given domain is specified, automatic subsumption becomes more useful.

Two noteworthy ontology acquisition projects, [LPS85] and Tacitus [Hob86], also bear significantly on our research. CYC shares with the ONTOS project the idea that a frame-based knowledge acquisition system holds promise for a qualitatively new kind of information system based on the multiply connected nature of knowledge (rather than linear text) and on the organization of material based on individual users, rather than on traditional article boundaries and sequencing. However, the knowledge acquisition methodology guiding the CYC project differs from the one employed by ONTOS in that the former:

- stresses the empirical side of this activity. Text analysis of randomly selected encyclopedia articles is not based on any prior domain analysis nor on any higher level ontological analysis.
- attempts to represent all of commonsense knowledge, rather than approaching this piecemeal, a domain at a time. In the ONTOS project the accent is on the immediate applicability of the knowledge acquisition tool in a variety of AI applications. We realize that some ontological distinctions will also come about bottom-up, through intensive domain analysis, but we recommend an intensive ontological or top-down analysis to guide the more bottom-up, empirical research.

The commitment to the higher levels of ontology is a feature that our approach shares with Tacitus, where a rich set of commonsense knowledge axioms has already been accumulated. What distinguishes our project is the goal of describing multiple applications and supporting multiple types of reasoning for large-scale applications. Tacitus is a theoretical exercise of much merit, but one totally independent of data

base considerations, issues of scaling up, multi-functionality of the knowledge, or any kind of knowledge acquisition or management system.

4.2. Acquisition Systems and Methodologies

The sheer amount and complexity of knowledge needed by computers to perform real-life tasks intelligently, including learning, requires intelligent human participation in the acquisition of this knowledge. However, by the same token, humans cannot accomplish this task without the aid of properly programmed computers. Knowledge acquisition for computer use is both a human-intensive and machine-intensive activity. Recognizing the situation for what it is, a number of AI researchers have strongly recommended that this process begin in earnest [LPS85] [NR87b]. The following is a brief survey of some of the preliminary steps taken.

The CYC project [LPS85] at MCC is aimed at constructing a massive data base of commonsense and more specialized information of encyclopedic proportions, in fact using the encyclopedia as a major source of knowledge. Unlike our project, CYC is by design empirical, its focus being predominantly on volume, with the hope that future uses will be found for this knowledge. We approach the problem in a completely different way, focusing on developing a sound semantics to ensure accuracy and permanence of the knowledge, attending to multiple applications of the knowledge base (reasoning systems in each of our domains) in order to validate both structure and content, and, most importantly, designing and building the knowledge representation language, management and acquisition system, and underlying distributed data base in order to support VLKB activity. We strive to develop a methodology and general purpose AI software to advance the science and technology so others can also build effective VLKBs – rather than building a single *ad hoc* exemplar.

Database technology became truly effective with the advent of the first general purpose DBMS and its foundational (if retrospectively simple) theory, rather than just depositing volumes of text in magnetic tape and hoping for the best. In this viewpoint we are more closely aligned with LUKE [Ric87], the new prototype natural language knowledge base management system also being developed at MCC.

Among the largest databases now stored by computers are information storage and retrieval systems. Systems such as DIALOG, MEDLARS, BRS, LEXIS, and WESTLAW each store tens of gigabytes of English text [SM83]. These systems use inverted files to provide rapid indexing of their texts. We believe the next step for information storage and retrieval systems is knowledge-based indexing and intelligent interactive search support (see [MC87] and discussion that follows).

Recently state-of-the-art user interfaces which enable communication through menus and direct manipulation of graphic objects have been integrated with AI programming environments [SBMC83,KKW84]. Some of these include knowledge-based editors [TW87,SS83]. However, browsers provided in these systems are very general and are not set up to provide intelligent aid for creating very large, semantically textured, domain representations with multiple uses.

State-of-the-art interfaces to more specific AI applications have emphasized viewing the knowledge base or retrieving information from it [TWF*82,RC85,PBL84,MC87], rather than knowledge acquisition. ONTOS has a knowledge-based interface that provides a graphic map of ontological categories and distinctions to guide entering knowledge for specific domains. This facility will be integrated with a natural language component that enables question-answering and learning by being told [HH83].

Question-answering systems do not merely retrieve text, but are capable of using it to answer questions. The XCALIBUR system [CBMA83], built at CMU, provides natural language query access to DEC's part

database for its VAX family of computers. It can answer questions such as 'What is the largest disk under $40,000?' The LADDER system [Hen77] answered questions about the position and status of ships at sea, for example 'To what country does the fastest sub belong?' Both of these systems are built on substantial knowledge bases, but both are special purpose KBs. The ASK [TT83a] and INTELLECT systems [Har79] have some similar capabilities.

The FERRET system [MCT87,Mau86], being built at CMU, is a general purpose information retrieval system built on the FrameKit [CJ85] knowledge representation language. FERRET reads news message and bulletin board posts from Dow Jones and UseNet News wires and uses a text skimming parser to generate instantiated sketchy scripts (stored as FrameKit frames) to represent the stories. These are matched to script patterns representing the user queries to retrieve relevant information with better precision than possible with simpler keyword systems. A high degree of precision is important for very large databases, because there may be hundreds or thousands of texts superficially related to the user query. One important limitation of FERRET is that it needs a very large KB itself in order to properly understand large numbers of texts.

However, a large-scale knowledge base will require tens of thousands of concepts for describing any serious application. It is clear that such an effort will be extremely labor-intensive. Thus, for instance, the experience of the 'Meaning - Text Model' group over the last two decades [MZ70,Mel74,Mel79], [Mel81,Mel82,Mel84,MZ84] and [AMZ69,Apr73] demonstrates the enormous cost of labor for producing their high-quality dictionary.

To significantly reduce this cost, a knowledge acquisition effort needs to draw significantly on the impressive interactive knowledge acquisition and user customization tools accumulated in the last decade, most recently by researchers in the field of transportable natural language interfaces, such as ASK [TT83b,TT85], INTELLECT [Har77], IRUS/IRACQ [BB83,Bat84,AW87], TQA [Dam78,Dam85], TEAM [GAMP87], LDC/TELI [BS86], LanguageCraft™ [Cel86], EUFID [TB83], CHAT-80 [WP82,Gin83]. Close to these systems is recent work on NIKL [KR86], providing a highly-developed KR environment, designed to support knowledge acquisition work. The ONTOS project is building on these past efforts, and extending them either in scope or semantic refinement.

References

[Ahl85] T.E. Ahlswede. A tool kit for lexicon building. In *Association for Computational Linguistics*, pages 268–276, 1985.

[AMZ69] Ju. D. Apresyan, I. A. Melv̌cuk, and A. K. Zholkovsky. Semantics and lexicography: towards a new type of unilingual dictionary. In F. Kiefer, editor, *Studies in Syntax and Semantics*, Dordrecht, 1969.

[Apr73] D. Ju Apresyan. *Trends in Soviet Theoretical Linguistics*. Reidel, Dordrecht, 1973.

[AW87] Shaked V. Ayuso, D.M. and R.M. Weischedel. An environment for acquiring semantic information. In *Proceedings of the ACL-87*, pages 32–40, Palo Alto, CA, 1987.

[Bat84] M. Bates. The irus transportable natural language interface. In *Proceedings of the First International Workshop on Expert Database Systems*, pages 258–274, 1984.

[BB83] M. Bates and R. Bobrow. A transportable natural language interface for information retrieval. In *Proceedings of the 6th International ACM SIGIR Conference*, 1983.

[BBC*87] B. Boguraev, T. Briscoe, J. Carroll, D. Carter, and C. Grover. The derivation of a grammatically indexed lexicon from the longman dictionary of contemporary english. In *Proceedings of ACL '87*, pages 193–200, 1987.

[BFL83] R. J. Brachman, R. E. Fikes, and Hector J. Levesque. Krypton: a functional approach to knowledge representation. *IEEE Computer*, 16:10, 1983.

[BL85] R. Brachman and H. Levesque. *Readings in Knowledge Representation*. Morgan Kaufmann, Los Altos, 1985.

[BS80] L. Birnbaum and M. Selfridge. Conceptual analysis in natural language. In *Inside Computer Understanding*, page 318, New Jersey: Erlbaum Assoc., 1980.

[BS85] R. Brachman and J. Schmolze. An overview of the kl-one knowledge representation system. *Cognitive Science*, 9, 1985.

[BS86] B.W. Ballard and D.E. Stumberger. Semantic acquisition in teli: a transportable, user-customized natural language processor. In *Proceedings of the ACL-86*, pages 20–29, 1986.

[BW77] D. G. Bobrow and T. Winograd. An overview of krl, a knowledge representation language. *Cognitive Science*, 1(1):3–46, 1977.

[Cal84a] N. Calzolari. Detecting patterns in a lexical database. In *Proceedings of COLING 1984*, pages 170–173, 1984.

[Cal84b] N. Calzolari. Machine-readable dictionaries, lexical database and the lexical system. In *Proceedings of COLING 1984*, page 460, 1984.

[Car81] J. G. Carbonell. *Subjective Understanding: Computer Models of Belief Systems*. Ann Arbor, MI: UMI research press, 1981.

[Car83] J. G. Carbonell. Learning by analogy: formulating and generalizing plans from past experience. In *Machine Learning, An Artificial Intelligence Approach*, Tioga Press, Palo Alto, CA, 1983.

[Car86] J. G. Carbonell. Derivational analogy: a theory of reconstructive problem solving and expertise acquisition. In *Machine Learning, An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, 1986.

[CB85] M. S. Chodorow and R. Byrd. Extracting semantic hierarchies from a large on-line dictionary. In *Proceedings of ACL '85*, pages 299–304, 1985.

[CBMA83] J. Carbonell, W. Boggs, M. Mauldin, and P. Anick. The xcalibur project, a natural language interface to expert systems. In *IJCAI83*, 1983.

[CCG81] J.G. Carbonell, R.E. Cullingford, and A.G. Gershman. Steps towards knowledge-based machine translation. 1981.

[Cel86] Monica Cellio. *The Language Craft Manual*. Carnegie Group, Inc., 1986.

[CG87] R. E. Cullingford and M. A. Graves. Knowledge acquisition for nlp: two prospects. In *Proceedings of the Natural Lake Planning Workshop*, page Paper 3, 1987.

[CH88] J. G. Carbonell and P. H. Hayes. Robust parsing using multiple construction-specific strategies. In L. Bolc, editor, *Natural Language Parsing Systems*, Springer-Verlag Publishers, 1988.

[CJ85] J. G. Carbonell and R. Joseph. The framekit reference manual. 1985. CMU Computer Science Department internal paper.

[Col60] A. Collin. The automatic construction of a glossary. *Information and Control*, IX, 1960.

[Cul86] R. E. Cullingford. *Natural Language Processing: A Knowledge Engineering Approach*. Rowman and Littlefield., Totowa, N.J., 1986.

[Dam78] F. Damereau. The derivation of answers from logical forms in a question answering system. *American Journal of Computational Linguistics*, 3–42, 1978.

[Dam85] F. Damerau. Problems and some solutions in customization of natural language database front ends. *ACM Transactions on Office Information Systems*, 3:2, 1985.

[DeJ79] G.F. DeJong. Prediction and substantiation: a new approach to natural language processing. *COGSCI*, 3, September 1979.

[Fas86] D. Fass. *Collative Semantics: A Description of the Meta5 Program*. Technical Report, Computer Research Laboratory, New Mexico State University, 1986.

[Fin79] N.V. Findler. *Associative Networks*. New York: Academic Press, 1979.

[For83] K. Forbus. Qualitative reasoning about space and time. In *Mental Models*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[FWS*86] E. A. Fox, R. C. Wohlwend, P. R. Sheldon, Q. F. Chen, and R. K. France. *Building the CODER Lexicon: The Collins English Dictionary and Its Adverb Definitions*. Technical Report, Virginia Polytechnical Institute and State University, 1986.

[GAMP87] B.J. Grosz, D.E. Appelt, P.A. Martin, and C.N. Pereira. Team: an experiment in the design of transportable natural-language interfaces. *Artificial Intelligence*, 32:173–243, 1987.

[Gin83] J. Ginsparg. A robust portable natural language data base interface. In *Conference on Applied Natural Language Processing*, pages 25–30, 1983.

[Gri70] J. E. Grimes. Computing and lexicography. *Linguistic Reporter*, V, 1970.

[Har77] L. Harris. User oriented data base query with the robot natural language query system. In *Proc. of the Third International Conference on Very Large Data Bases*, Oct 1977.

[Har79] L. Harris. Experience with natural language database query applications. pages 365–368, 1979.

[Hen77] G. G. Hendrix. *The LIFER Manual: A guide to Building Practical Natural Language Interfaces.* Technical Report Tech. note 138, SRI, 1977.

[HH83] N. Haas and G. G. Hendrix. Learning by being told: acquiring knowledge for information management. In *Machine Learning, An Artificial Intelligence Approach*, Tioga Press, Palo Alto, CA, 1983.

[Hob5a] J.R. Hobbs. Granularity. In *Proceedings of IJCAI '85*, pages 432–435, 1985a.

[Hob85] J. R. Hobbs. Introduction. In J.R. Hobbs and R. C. Moore, editors, *Formal Theories of the Commonsense World*, Ablex, 1985.

[Hob86] J. R. Hobbs. Overview of the tacitus project. *Computational Linguistics*, 12:3, 1986.

[Hol75] J. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[Jac85] P. Jacobs. Phred: a generator for natural language interfaces. *Computational Linguistics*, 11:4, 1985.

[KKW84] J.C. Kunz, T.P. Kehler, and M.D. Williams. Applications development using a hybrid ai development system. *AI Magazine*, 5(3):41–54, Fall 1984.

[Kol84] J. L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Erlbaum, Hillsdale, N.J., 1984.

[KR86] Bates R. Kaczmarek, T.S. and G. Robins. Recent developments in nikl. In *Proceedings AAAI-86*, Philadelphia, PA, 1986.

[Kuc69] H. Kucera. Computers in language analysis and lexicography. In *American Heritage Dictionary of the English Language*, Houghton Mifflin, 1969.

[LB85] H.J. Levesque and R. J. Brachman. A fundamental tradeoff in knowledge representation and reasoning (revised version. In R.J. Brachman and H.L. Levesque, editors, *Readings in Knowledge Representation*, Morgan Kaufmann, 1985.

[Leb85] M. Lebowitz. *An Experiment in Intelligent Information Systems: RESEARCHER.* Technical Report, Columbia University, Dept. of Computer Science, Nov 1985.

[LPS85] D. Lenat, M. Prakash, and M. Shepherd. Cyc: using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine*, VI:65–85, 1985.

[Mau86] M.L. Mauldin. Retrieving information with a text skimming parser. In *Proceedings of the 1986 Eastern States Conference on Linguistics*, October 1986.

[MC87] I. Monarch and J. G. Carbonell. Coalsort: a knowledge-based interface. *IEEE Expert*, 2(1):39–53, 1987.

[MCT87] M.L. Mauldin, J.G. Carbonell, and R.H. Thomason. Beyond the keyword barrier: knowledge-based information retrieval. In *Proceedings of the 29th Annual Conference of the National Federation of Abstracting and Information Services*, North-Holland, March 1987.

[Mel74] I. A. Melvcuk. *Towards a Theory of Linguistic Models of the Meaning-Text Type.* Nauka, Moscow, 1974.

[Mel79] I. A. Melvcuk. *Studies in Dependency Syntax.* Karoma, Ann Arbor, 1979.

[Mel81] I. A. Melvcuk. Meaning-text models: a recent trend in soviet linguistics. *Annual Review of Anthropology*, 10, 1981.

[Mel82] I. A. Melv̌cuk. Lexical functions in lexicographic description. In *Proceedings of the 8th Annual Meeting of the Berkeley Linguistics Society.*, pages 427–444, 1982.

[Mel84] I. A. Melv̌cuk. *Dictionnaire explicatif et combinatoire du francais contemporain.* University of Montreal Press., Montreal, 1984.

[MZ70] I. A. Melv̌cuk and A. K. Zholkovsky. Towards a functioning 'meaning-text' model of language. 57, 1970.

[MZ84] I. A. Melv̌cuk and A. K. Zholkovsky. *Explanatory Combinatorial Dictionary of Modern Russian.* Wiener Slawistischer Almanach, Vienna, 1984.

[NC87] S. Nirenburg and J. Carbonell. Integrating discourse pragmatics and propositional knowledge in multilingual natural language processing. *Computers and Translation*, 2:105 – 116, 1987.

[Nir85] S. Nirenburg. *Proceedings of the Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages.* Colgate University, 1985.

[Nir88] S. Nirenburg Lexicographic support for knowledge-based machine translation. In A. Zampolli, editor, *Computational lexicology and lexicography*, Pisa, Italy, Giardini, 1988.

[NR86] S. Nirenburg and V. Raskin. A metric for computational analysis of meaning: toward an applied theory of linguistic semantics. In *Proceedings of COLING-86*, pages 338–340, 1986.

[NR87a] S. Nirenburg and V. Raskin. The analysis lexicon and the lexicon management system. *Computers and Translation*, 2:177 – 188, 1987.

[NR87b] S. Nirenburg and V. Raskin The subworld concept lexicon and the knowledge base Management System. *Computational Linguistics*, 13:3, 1987.

[PBL84] P.F. Patel-Schneider, R.J. Brachman, and H.J. Levesque. *Argon: Knowledge Representation meets Information Retrieval.* Technical Report 654, September 1984.

[RC85] M.H. Richer and W.J. Clancey. *Guidon-Watch: A Graphics Interface for Viewing A Knowledge Base.* Technical Report, Stanford Knowledge Systems Laboratory, 1985.

[Ric87] E. Rich. Luke: a kbms. In *Proceedings of Applied ACL-87*, 1987.

[SA77] R. C. Schank and R. P. Abelson. *Scripts, Goals, Plans and Understanding.* Hillside, NJ: Lawrence Erlbaum, 1977.

[SBMC83] M. Stefic, D. Bobrow, S. Mittal, and L. Conway. Knowledge programming in loops: report on an experimental course. *AI Magazine*, IV(3):3–13, Fall 1983.

[Sch75] R. C. Schank. *Conceptual Information Processing.* Amsterdam: North-Holland, 1975.

[Sch82] R. C. Schank. *Dynamic Memory.* Cambridge University Press, 1982.

[She87] P. Shell. Parmenides: a reference manual. 1987. CMU Working Document.

[SM83] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval. McGraw-Hill Computer Science Series*, McGraw-Hill, New York, 1983.

[SPB88] A. Z. Spector, R. F. Pausch, and G. Bruell. Camelot: a flexible, distributed transaction processing system. In *Proceedings of Compcon 88*, 1988.

[SS83] E. Schoen and R.G. Smith. Impulse, a display-oriented editor for strobe. In *Proceedings of the National Conference on AI*, pages 356–358, 1983.

[SS87] A. Z. Spector and K. R. Swedlow. *The Guide to the Camelot Distributed Transaction Facility: Release 1*. Carnegie Mellon University, Pittsburgh, PA, 0.7(33a) edition, 1987.

[TB83] M. Templeton and J. Burger. Problems in natural language interface to dbms with examples from eufid. In *Conference on Applied Natural Language Processing*, pages 3–16, 1983.

[TT83a] B. Thompson and F. Thompson. Introducing ask, a simple knowledgeable system. In *Conference on Applied Natural Language Processing*, pages 17–24, 1983.

[TT83b] B.H. Thompson and F.B. Thompson. Introducing ask, a simple knowledgeable system. In *AAAI Tutorial on Natural Language Processing*, AAAI, Aug 1983.

[TT85] B. Thompson and F. Thompson. Ask is transportable in half a dozen ways. *ACM Transactions on Office Information Systems*, 3:2, 1985.

[Tul85] E. Tulving. How many memory systems are there? *American Psychologist*, 40, 1985.

[TW87] W.F. Tichy and B. Ward. *A Knowledge-Based Graphical Editor*. Technical Report, Universitat Karlsruhe, Fakultat Fur Informatik, Jan 1987.

[TWF*82] F. Tou, M. Williams, R. Fikes, A. Henderson, and T. Malone. Rabbit: an intelligent database assistant. In *Proceedings of the National Conference on Artificial Intelligence*, pages 314–318, 1982.

[Ven73] R. L. Venezky. Computational aids to dictionary compilation. In R. Frank and Cameron A. F., editors, *A Plan for the Dictionary of Old English*, University of Toronto Press, 1973.

[Wal84] D. E. Walker. Machine-readable dictionaries. In *Proceedings of COLING '84*, page 457, 1984.

[Wal87] D. E. Walker. Knowledge resource tools for accessing large text files. In S. Nirenburg, editor, *Machine Translation: Theoretical and Methodological Issues*, Cambridge University Press, 1987.

[WF84] Y. A. Wilks and D. Fass. Preference semantics, ill-formedness and metaphor. *American Journal of Computational Linguistics*, 9:2, 1984.

[WFG*87] Y. A. Wilks, D. Fass, C. M. Guo, J. E. M. McDonald, T. Plate, and B. M. Slator. A tractable machine dictionary as a resource for computational semantics. In *Proceedings of the Natural Language Workshop in Blue Mountain Lake*, page Paper 6, 1987.

[Wil83] R. Wilensky. *Planning and Understanding*. Addison Wesley, Reading, MA, 1983.

[Wil84] R. Wilensky. Kodiak - a knowledge representation language. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, pages 344–352, 1984.

[WP82] D. H. D. Warren and F. C. N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8:3-4, 1982.

[YH85] S.R. Young and P.J. Hayes. Tess: a telex classifying and summarization system. In *The Second Conference on Artificial Intelligence Applications*, IEEE, 1985.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November 1988

INTEGRATION OF ACQUISITION AND PERFORMANCE

Alain T Rappaport and Brian R Gaines

Neuron Data, 444 High St, Palo Alto, CA 94301
and
Knowledge Science Institute, University of Calgary
Calgary, Alberta, Canada T2N 1N4

ABSTRACT

Knowledge base building environments must progress in two important directions: (i) increased participation of domain experts in the knowledge design process through new computational models and effective man-machine interfaces and (ii) automated knowledge acquisition tools to facilitate the overt expression of knowledge. This paper presents the integration of a knowledge acquisition methodology with a performance system. The resulting architecture represents a combination of techniques from psychology, cognitive sciences and artificial intelligence. New dimensions emerge from this implementation and integration both at the theoretical and practical levels. The overall system is not linked to a particular control structure and is not task dependent. We discuss the value of intermediate representations in this context and the role of different approaches to the induction process. Topological induction is particularly efficient in the elicitation process and stresses the importance of interactive inductive techniques with participation from the experts. While the knowledge acquisition tool provides an analysis and structuring of the domain knowledge, the control is implemented using the performance system's interface. Therefore, both modules participate in the overall knowledge acquisition process. Beyond the integration of these knowledge acquisition and performance systems, the architecture can also be integrated with databases, text analysis techniques, and hypermedia systems.

INTRODUCTION

The integration of knowledge acquisition tools and performance systems is a highly significant issue in knowledge-based system design. Achieving this integration raises problems of representational shift, domain and the task-dependence, knowledge validation and maintenance, and others. It brings to light new issues for AI research. While research continues on new representations and inference mechanisms, achieving such integration will provide new insights on how the techniques from the knowledge acquisition and machine learning communities relate to those involved in reasoning architectures. To make progress in integration, it is necessary to provide the techniques in a highly useable form which can be successfully implemented in real world environments. In the present paper, we explore some of these issues through the integration of methodologies based on the NEXPERT *OBJECT* and NEXTRA tools.

Figure 1 illustrates the knowledge acquisition process for current systems. The system developer draws upon three major knowledge sources: knowledge encoded in *relevant media*, such as books, journals and videotapes; knowledge available by observing and modeling the *relevant domain*; and knowledge available from discourse with, and observation of, the *relevant community*. It is this last source that is most associated with expert systems as such, although all three sources play significant roles in the development of any practical system.

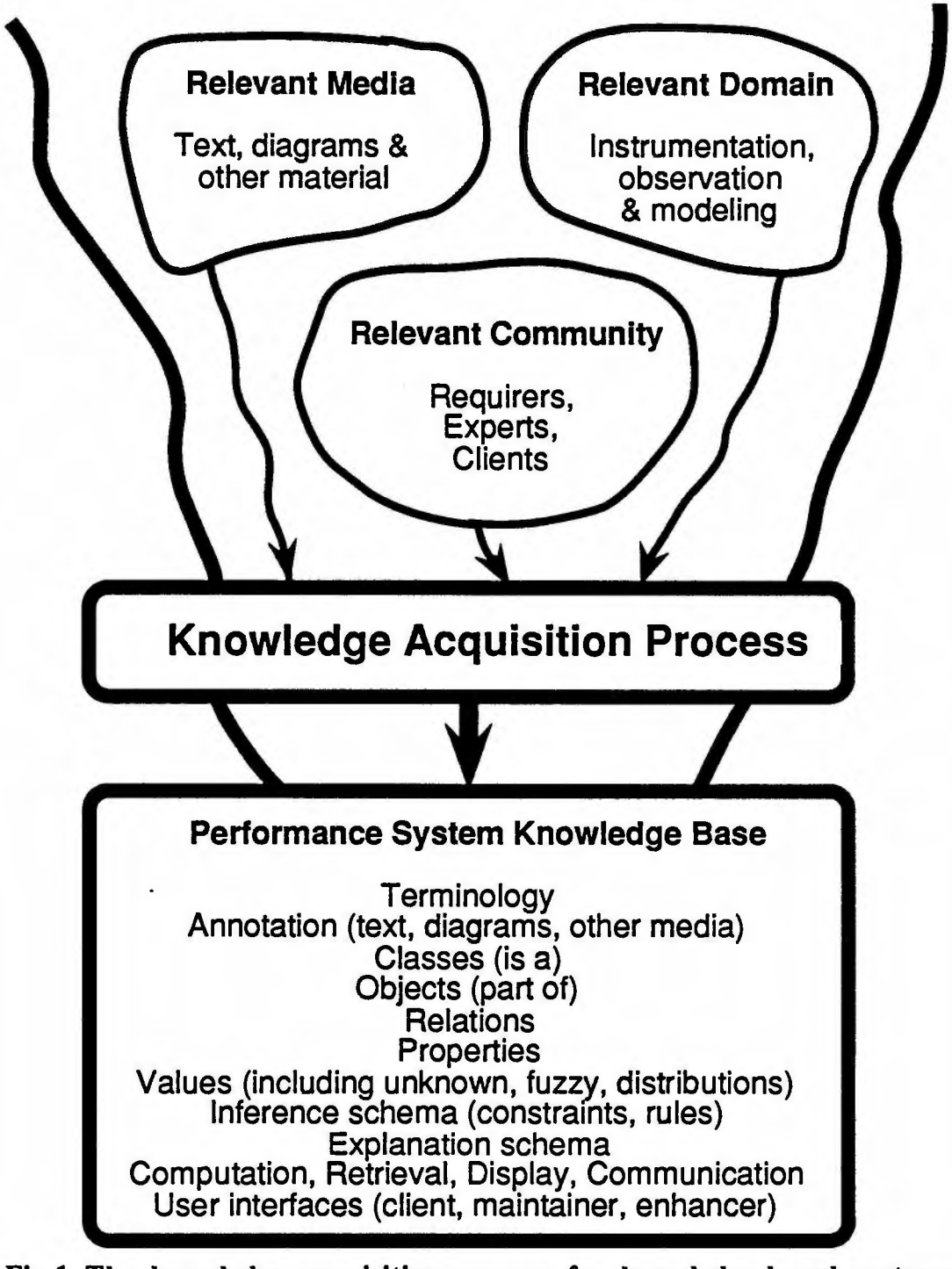


Fig.1 The knowledge acquisition process for knowledge-based systems

Note that the *relevant community* has been split into requirers, experts and clients. Domain experts are the expected source of knowledge for an expert system, but detailed requirements specifications and involvement of end-users play as a significant role as they do in other computing system design. It is useful to think of those with these different roles as being all 'experts' in some aspect of the system functioning, since the knowledge acquisition techniques used with domain experts are also well-suited to establishing requirements specifications and end-user perspectives.

The content of the *performance system knowledge base* in Figure 1 illustrates the range of technical features which an adequate knowledge-based system shell may be expected to support:

- *Terminology* is often neglected as computationally significant. However, it is particularly important in knowledge-based systems where it is critically important that the terms used in questions, recommendations and explanations are understood by clients in the same way as they are by experts. Obtaining the terminology direct from experts without distortion is an important design consideration for knowledge acquisition systems.
- *Annotation* is used here as a term for non-computational knowledge such as text, diagrams, photographs, videos, and other media, that may be used to impart knowledge to people. Some of it may not be suitable for computer storage but not for inferential processing. Providing appropriate access to such knowledge is important in many knowledge-based systems, and collecting and organizing such knowledge is part of the knowledge acquisition process.
- *Classes, objects, relations, properties* and *values* form the abstract infrastructure of computational knowledge representation currently. Object-orientated knowledge-bases provide a formal conceptual and computational schema which subsumes and supports a wide variety of knowledge representation schema (Shriver & Wegner 1987). These primitives also correspond to natural cognitive operations of classifying, instantiating, relating, attributing and measuring. An important aspect of knowledge acquisition is to determine these cognitive primitives in a domain (Rappaport 1987b).

  Where knowledge representation goes beyond conventional data representation is largely in the variety of types of value that must be supported. In a data base, a numeric field may be expected to have a clearly defined value. In a knowledge base, because knowledge is 'less than truth', it may be necessary to allow for the value being unknown, constrained by a fuzzy hedge, a range, a distribution, or some combination of these. There may also be mutual constraints between such ill-defined values.
- *Inference schema* are systems of constraints that express the relations between values in a knowledge base. These constraints may be thought of as rules expressing the relations and many of them may necessarily be expressed by explicit rules in the knowledge-base. However, the effective organization of the knowledge base in terms of generic classes and relationships should allow the majority of the constraints to be expressed implicitly in a natural way. This is the *framing problem* in knowledge representation for a domain, and an important aspect of knowledge acquisition is to determine a representation schema that minimizes the number of low level rules required.
- *Explanation schema* are forms of argument that generate satisfactory explanations of the inferences made from a knowledge base. Early expert systems generated explanations by displaying the chain of reasoning that was used from premises to conclusions. This approach is useful but does not necessarily 'explain' the conclusions in a way that an expert would, or in a way satisfying a client's need for knowledge. The acquisition of knowledge necessary for adequate explanation goes beyond that adequate for performance.
- *Computation, retrieval, display, communication* and other data processing activities play major roles in most knowledge-based systems. Modern expert system shells make provision for integration with data-processing, simulation, database, graphics and communications programs. The systems analysis for these needs to be integrated with the knowledge acquisition process.
- User interfaces are very important in knowledge-based systems as they are generally highly interactive and involve clients with little or no computer experience. The design of the interface in consultation with experts and clients is an important part of knowledge acquisition. Since most knowledge is fairly volatile, knowledge acquisition and updating the knowledge base are important aspects of system operation, and the interface to system maintainers and enhancers also needs careful attention.

NEXPERT AND NEXTRA

In our integrated knowledge support system the knowledge acquisition process is carried out by NEXTRA which creates a knowledge base for use by the performance system NEXPERT. One of our objectives is to create even closer integration by having the arrow from acquisition to performance be reversible, so that the knowledge base itself is always open to further development using the acquisition tools, even when it has been edited in the support environment of the performance tool. However, Figure 1 accurately represents the current state of the art in which acquisition takes place as a separate step prior to performance.

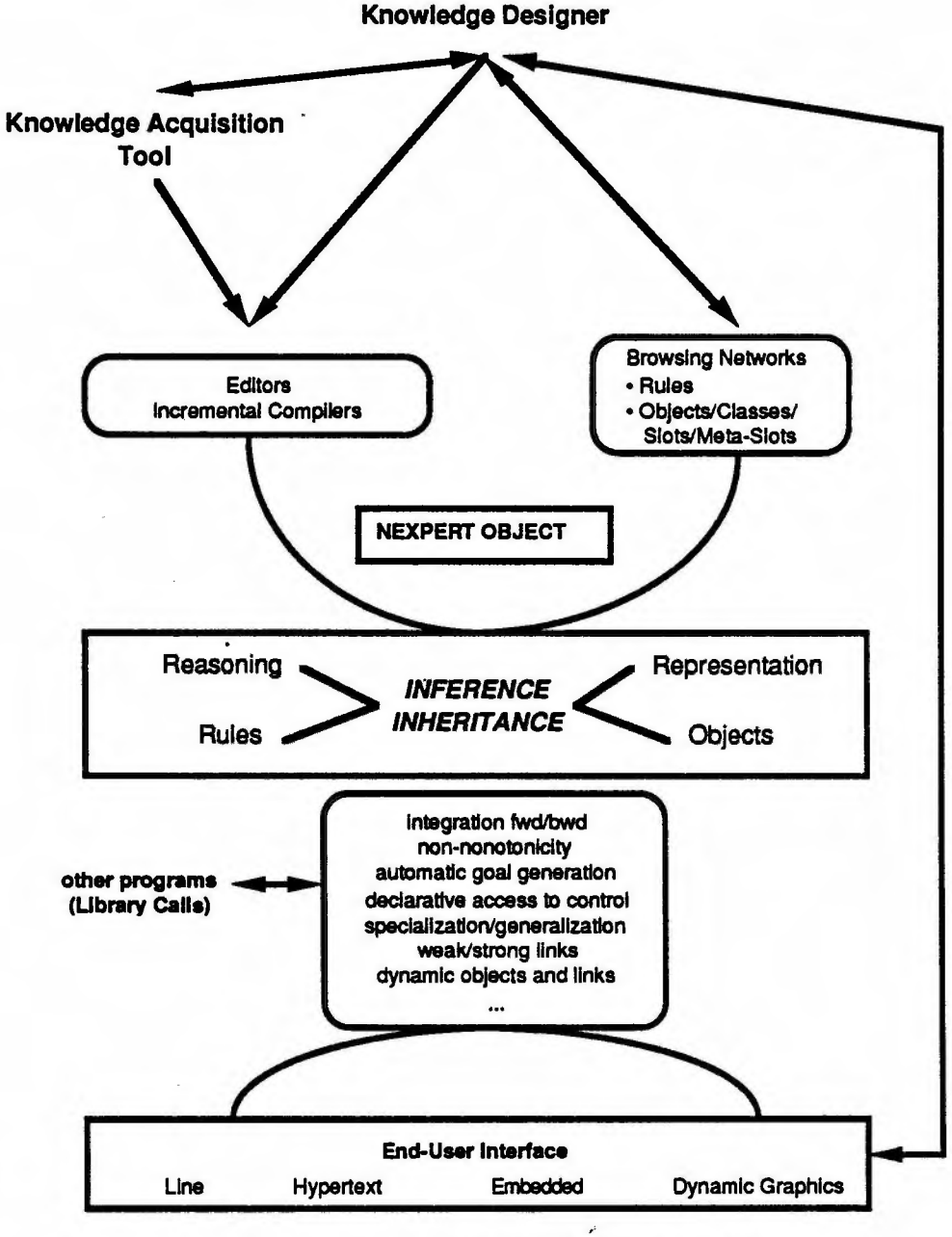


Fig.2 Architecture of NEXPERT *OBJECT*

NEXPERT *OBJECT* is a general-purpose development environment for the building of knowledge based systems with the architecture shown in Figure 2. Applications cover a wide range of tasks, from analytic ones such as interpretation and classification to synthetic tasks such as scheduling and planning. The generality of the tool results from an approach based on "cognitive primitives" (Rappaport, 1987b). These allow a great flexibility by permitting the developer or domain expert to build the control structure in terms of comprehensible primitives. This approach contrasts with earlier approaches based on the separation of control from knowledge. This would be ideal is the control structures were ideal. In most cases, however, when a system follows this separation, the restrictions imposed by the control structure make it task-dependent. Another fundamental aspect of the NEXPERT performance system is the highly graphic interface to visualize the reasoning and representation dimensions of the knowledge bases (Rappaport, 1987a).

NEXTRA is a knowledge acquisition toolbox based on an extensible set of techniques developed for the knowledge support system KSS0 (Gaines 1987a). The NEXTRA technology encompasses elicitation tools, visual analysis and display tools, group comparison tools, inductive tools, and knowledge base generative tools as shown in Figure 3.

The interviewing tools have their foundations in the personal construct psychology of Kelly (1955), subsequent work on the computer elicitation of conceptual structures through repertory grids (Shaw 1980), and applications of these techniques to knowledge elicitation (Boose 1984, 1986, Boose & Bradshaw 1986, Shaw & Gaines 1983, 1986, 1987a,b). The objective is to gather domain knowledge from prototypical entities represented along certain dimensions:

- **Interview** accepts specifications of entities within a sub-domain and provides an interactive graphical elicitation environment within which the experts can distinguish entities to derive their attributes. The resultant class is continuously analyzed to provide feedback prompting the expert to enter further entities and attributes.

The visual analysis tools consist of an interactive interface to represent the abstractions derived from those entities in terms of hierarchical clusters using Shaw's (1980) FOCUS algorithm, and relational diagrams such as a non-hierarchical conceptual maps derived through principal components analysis (Slater 1976). The objectives are to validate the raw domain knowledge and suggest further structure at a higher level through interactive topological induction:

- **Cluster** hierarchically clusters entities and attributes within a sub-domain prompting the experts to add higher-level entities structuring the domain.
- **Map** spatially clusters entities and attributes within a sub-domain prompting the experts to add higher-level entities structuring the domain.

The group comparison tools consist of an interactive interface to represent the relations between the terminologies and conceptual systems of different experts, or experts and clients. The objectives are to determine the consensus, conflict, correspondence and contrast between different conceptual systems (Shaw & Gaines 1988):

- **Compare** compares the structures for the same sub-domain generated by different experts, or the same expert at different times or from varying perspectives.

The inductive part consists in the derivation of constraints within the conceptual structures through logical entailment analyses (Gaines and Shaw 1985, Quinlan 1987, Cendrowska 1987). The objective is to suggest further structure at a higher level that translates into class inclusions or rules in NEXPERT:

- **Entail and Induct** induce logical entailments enabling the attributes of an entity, or the evaluations of a decision-making situation in a domain, to be derived from other attributes.

The generative part consists in the transformation of the knowledge analysis made by the previous tools into a formalism understandable by the NEXPERT inference mechanisms:

- **Nexport** formats the specifications of sub-domains as classes, of entities as objects, of attributes as properties, and of entailments as methods, and transfers them to the performance tool.

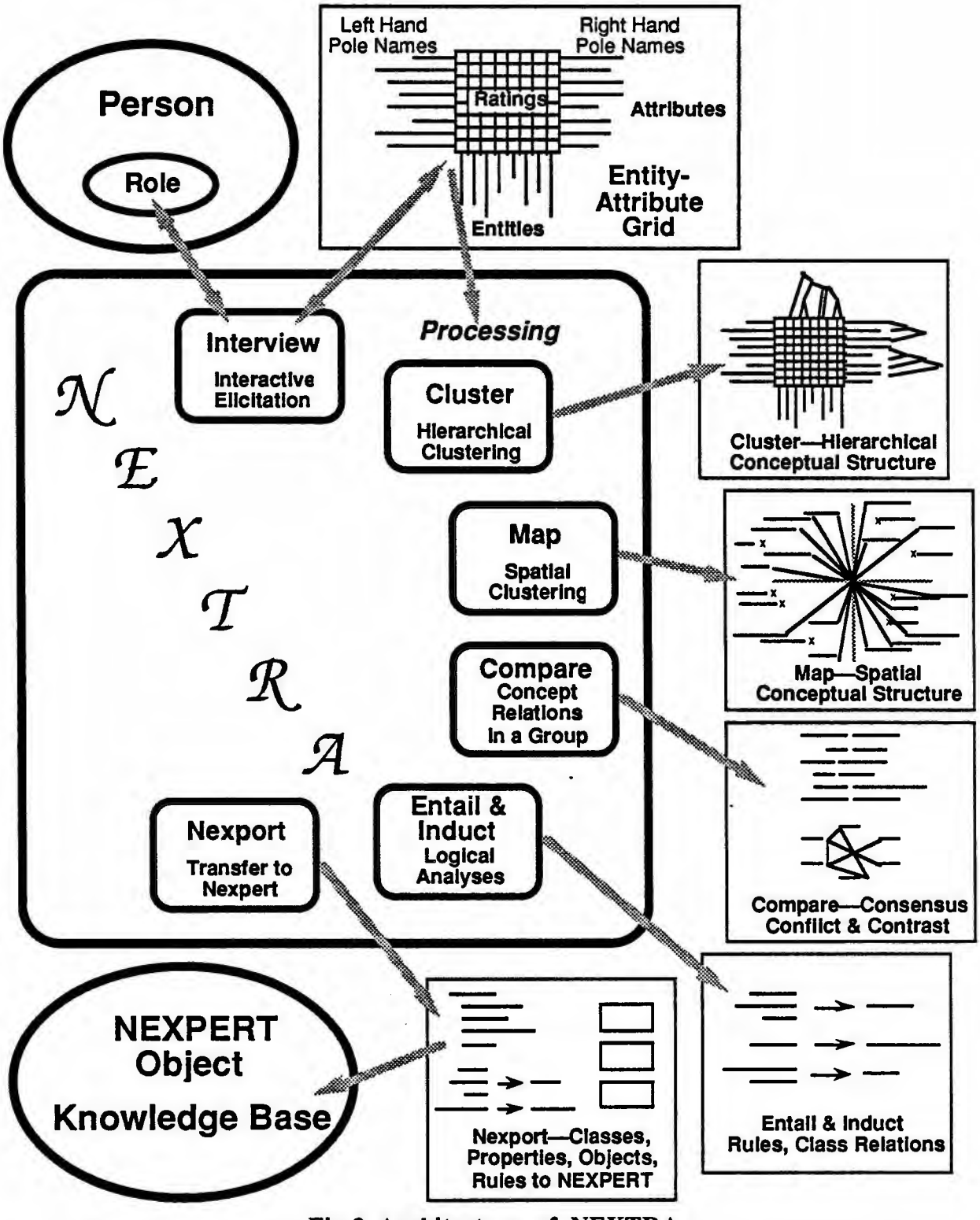


Fig.3 Architecture of NEXTRA

KNOWLEDGE CREATION, EDITING AND TRANSFER

A major aspect of the integration of NEXTRA and NEXPERT lies in the variety of debugging cycles due to the multiplicity of representations available. Whether the knowledge entry comes from elicitation, interactive topological induction, or an inductive algorithm, the immediate transfer into the performance systems allows the validation of knowledge at any stage of development. In fact, as shown in Figure 4, two general ways of editing or creating knowledge are now available: (i) through the knowledge acquisition tools and (ii) through the performance system development.

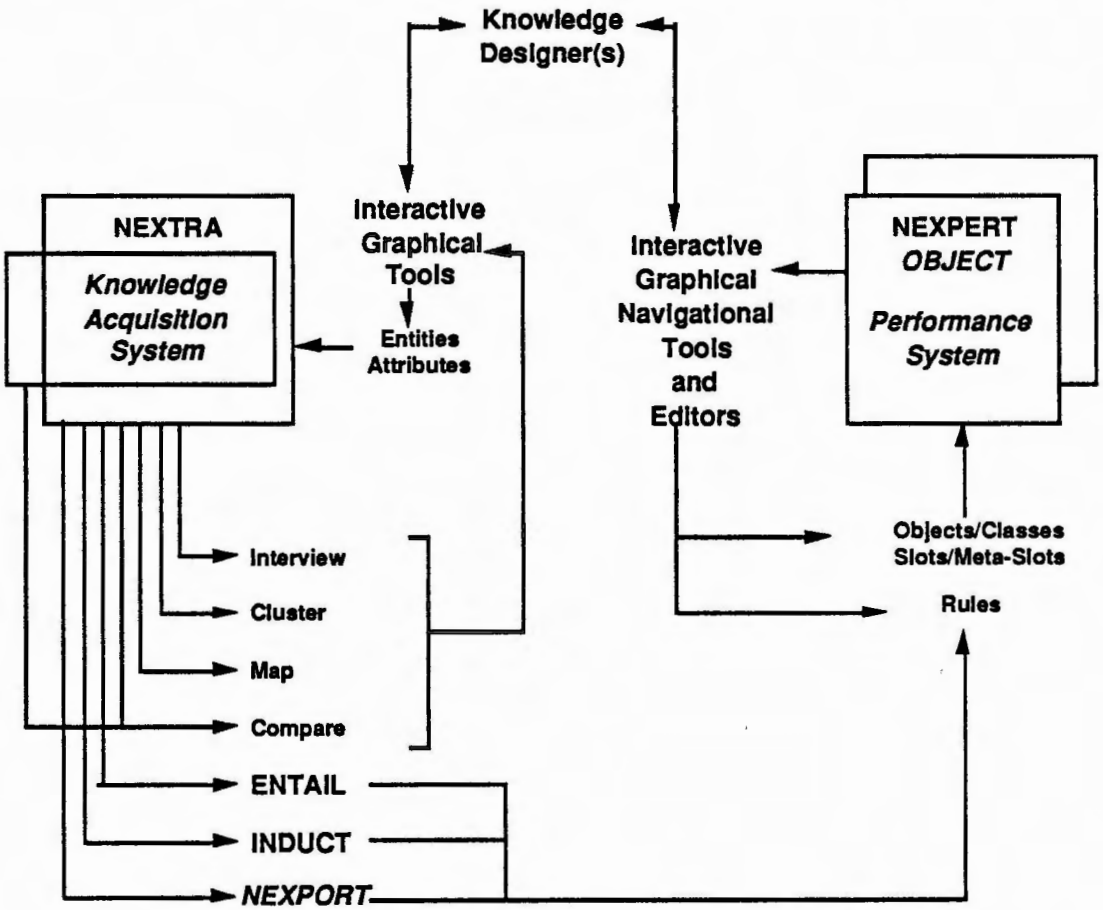


Fig.4 Integration of two knowledge creation and editing systems

The **Interview** tool allows entities and attributes to be specified through an interactive click and drag dialog as shown in Figure 5. The **Cluster** tool sorts the entities and attributes to produce a hierarchical cluster as shown in Figure 6. The **Map** tool arranges the entities and attributes to produce a spatial cluster as shown in Figure 7. The **Compare** tool matches entities and attributes between grids to produce correspondences as shown in Figure 8. The **Induct** tool derives entailments between attributes as shown in Figure 9. The **Nexport** tool converts attribute sets to class definitions, entities to objects and entailments to rules and formats them as a NEXPERT knowledge base as shown in Figure 10. The classes, objects and rules can then be edited in the NEXPERT graphic editing environment as shown in Figure 11. A typical expert system development will involve a number of elicitations with NEXTRA to develop the conceptual structures for the sub-domains followed by the use of NEXPERT editing tools to link the sub-domains through appropriate control structures.

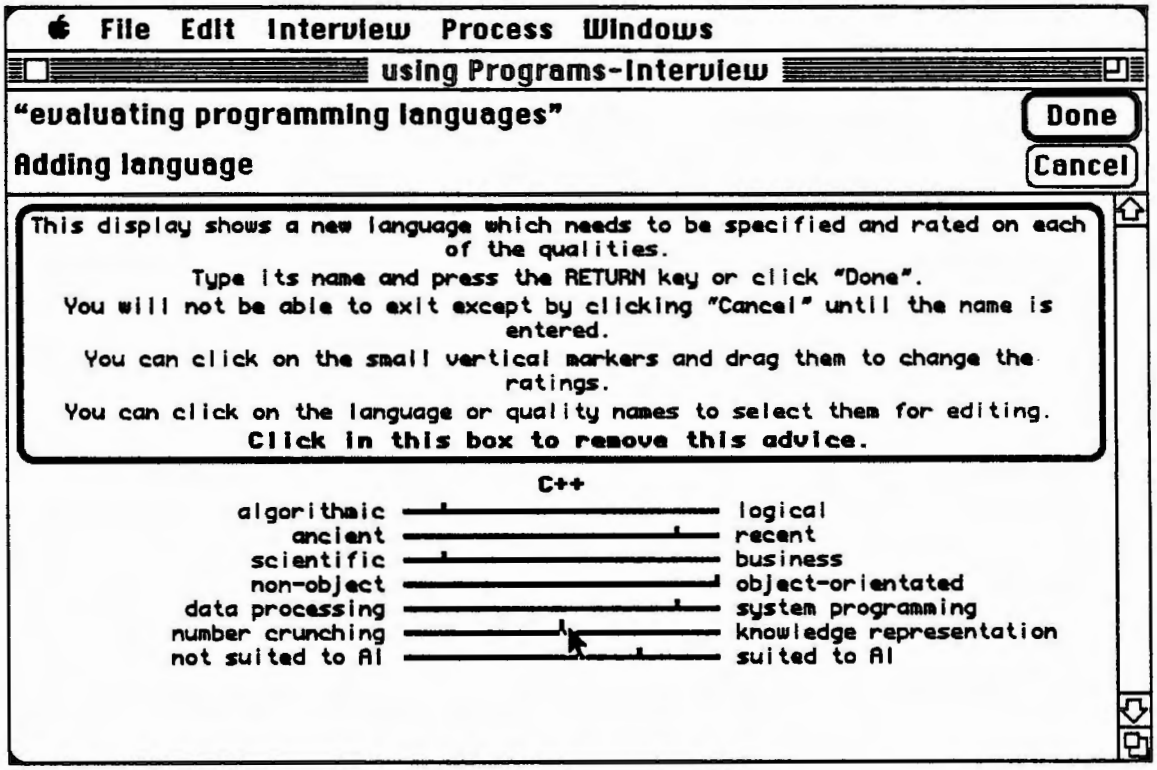


Fig.5 Interview entity rating screen

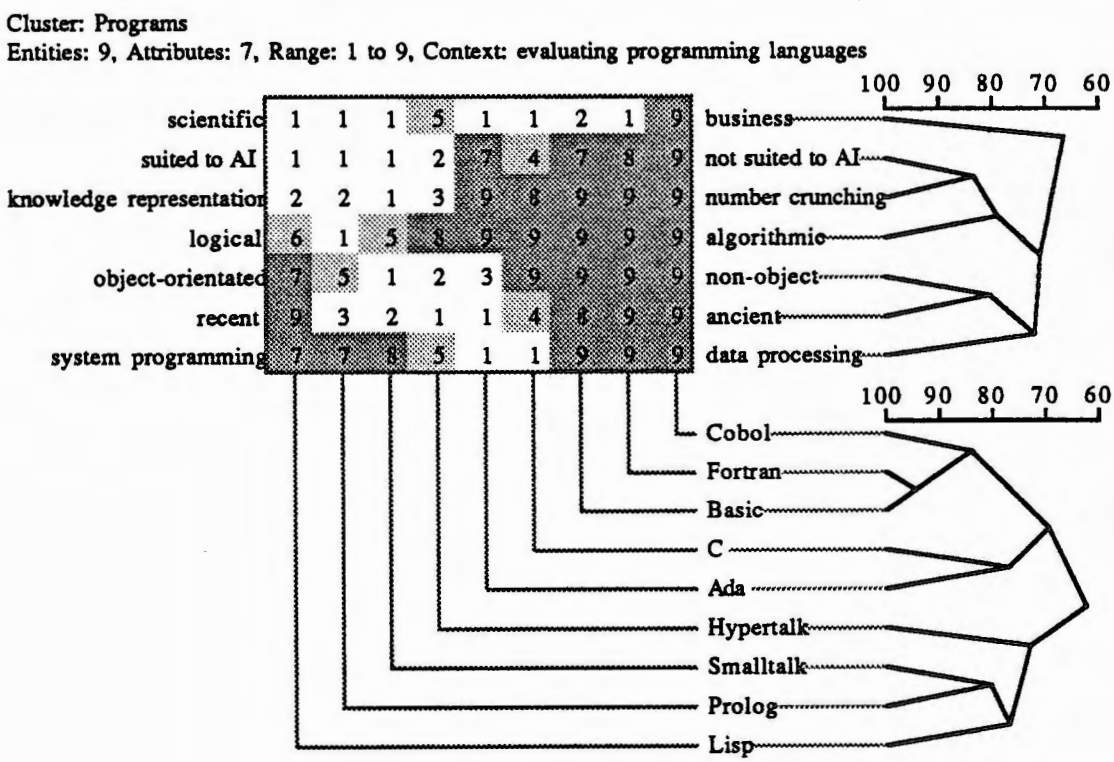


Fig.6 Cluster hierarchical display

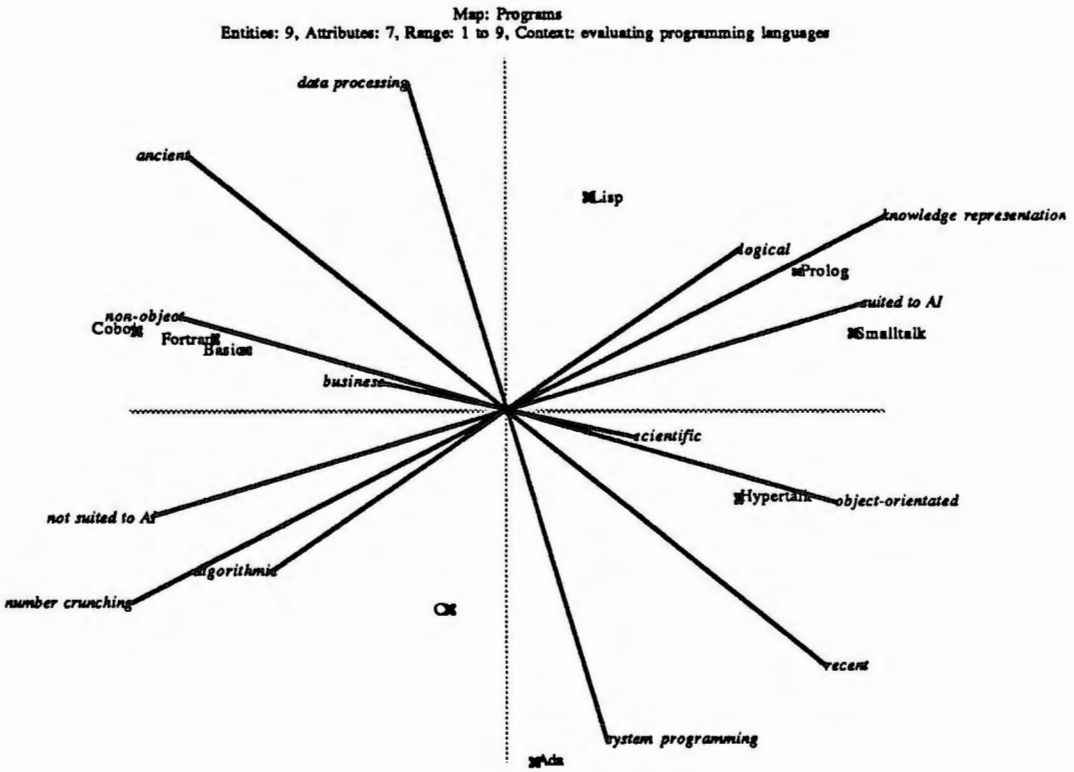


Fig.7 Map spatial display

```
1:   6.2% > 88.5    G1A2: Local - Global
                    G2A2: local - global
2:  12.5% > 87.5    G1A3: Low level data - High level data
                    G2A8:   nominal data - interval or ratio data
3:  18.8% > 86.5    G1A1: Does not honour data points - Honours data points
                    G2A4:  doesn't honour data points - honours data points
4:  25.0% > 86.5    G1A7: Short distance autocorrelation - Long distance autocorrelation
                    G2A2:                          local - global
5:  31.2% > 86.5    G1A9: New geographical technique - Old geographical technique
                   G2A18:            not widely used - widely used
6:  37.5% > 85.4   G1A16: Not widely used - Widely used
                   G2A18: not widely used - widely used
7:  43.8% > 83.3    G1A5: Discontinuous - Continuous
                    G2A2:         local - global
8:  50.0% > 82.3    G1A4: Mathematically complex - Mathematically simple
                   G2A11:   heavy computing load - no computing load
9:  56.2% > 81.2   G1A10:   Hard to adapt to multivariate - Easy to adapt to multivariate
                    G2A5: usually one variable considered - multiple variables considered
10: 62.5% > 81.2   G1A12:   Does not require spatial search - Requires spatial search
                   G2A13: estimates susceptible to clusters - not as susceptible to clusters
11: 68.8% > 79.2    G1A6: Does not require a priori model - Requires a priori model
                    G2A2:                          local - global
12: 75.0% > 79.2   G1A11:      Few points - Many points
                   G2A18: not widely used - widely used
13: 81.2% > 76.0   G1A13:          Does not use polynomial - Uses polynomial
                    G2A6: doesn't fit a mathematical curve - mathematical curve fitting
14: 87.5% > 76.0   G1A15: Not very effective - Very effective
                   G2A17: not very effective - very effective
15: 93.8% > 72.9   G1A14: Not very important - Very important
                   G2A18:    not widely used - widely used
```

Fig.8 Compare attribute correspondence display

```
recent≥3 & non-object≥2 -> logical≥3 (1/9:1/1 88.89)
non-object≥3 -> ancient≥2 (5/9:5/5 94.71)
non-object≥3 & data processing≥2 -> ancient≥3 (4/9:4/4 96.10)
object-orientated≥2 -> recent≥2 (5/9:4/4 90.47)
object-orientated≥2 -> recent≥3 (4/9:4/4 96.10)
ancient≥2 -> non-object≥3 (5/9:5/5 94.71)
recent≥3 -> object-orientated≥2 (4/9:4/4 96.10)
algorithmic≥2 & recent≥3 -> object-orientated≥3 (3/9:3/3 96.30)
algorithmic≥3 & recent≥2 -> system programming≥2 (3/9:3/3 96.30)
recent≥2 & number crunching≥2 -> system programming≥3 (2/9:2/2 95.06)
not suited to AI≥2 -> number crunching≥3 (5/9:5/5 94.71)
suited to AI≥3 -> knowledge representation≥3 (4/9:4/4 96.10)
number crunching≥2 -> not suited to AI≥2 (5/9:5/5 94.71)
ancient≥3 & number crunching≥2 -> not suited to AI≥3 (4/9:3/3 91.22)
algorithmic≥3 & ancient≥3 -> not suited to AI≥3 (4/9:3/3 91.22)
algorithmic≥3 & data processing≥3 -> not suited to AI≥3 (4/9:3/3 91.22)
knowledge representation≥2 -> suited to AI≥3 (4/9:4/4 96.10)
```

Fig.9 Induct entailments between attributes

```
(@PROPERTY=       algorithmic    @TYPE=Boolean;)
(@PROPERTY=       logical        @TYPE=Boolean;)
(@PROPERTY=       ancient        @TYPE=Boolean;)
(@PROPERTY=       recent @TYPE=Boolean;)
(@PROPERTY=       non_object     @TYPE=Boolean;)
(@PROPERTY=       object_orientated     @TYPE=Boolean;)
(@PROPERTY=       data_processing       @TYPE=Boolean;)
(@PROPERTY=       system_programming    @TYPE=Boolean;)
(@PROPERTY=       number_crunching      @TYPE=Boolean;)
(@PROPERTY=       knowledge_representation     @TYPE=Boolean;)
(@PROPERTY=       not_suited_to_AI      @TYPE=Boolean;)
(@PROPERTY=       suited_to_AI   @TYPE=Boolean;)

(@CLASS=   languages
    (@PROPERTIES=
           algorithmic
           logical
           ancient
           recent
           non_object
           object_orientated
           data_processing
           system_programming
           number_crunching
           knowledge_representation
           not_suited_to_AI
           suited_to_AI
    )
)

(@RULE=    Ind3
    (@LHS=
           (Yes    (<languages>.algorithmic))
           (Yes    (<languages>.data_processing))
    )
    (@HYPO=       not_suited_to_AI)
    (@RHS=
           (Let    (<languages>.not_suited_to_AI)      (True))
    )
)
```

Fig.10 Nexport knowledge base

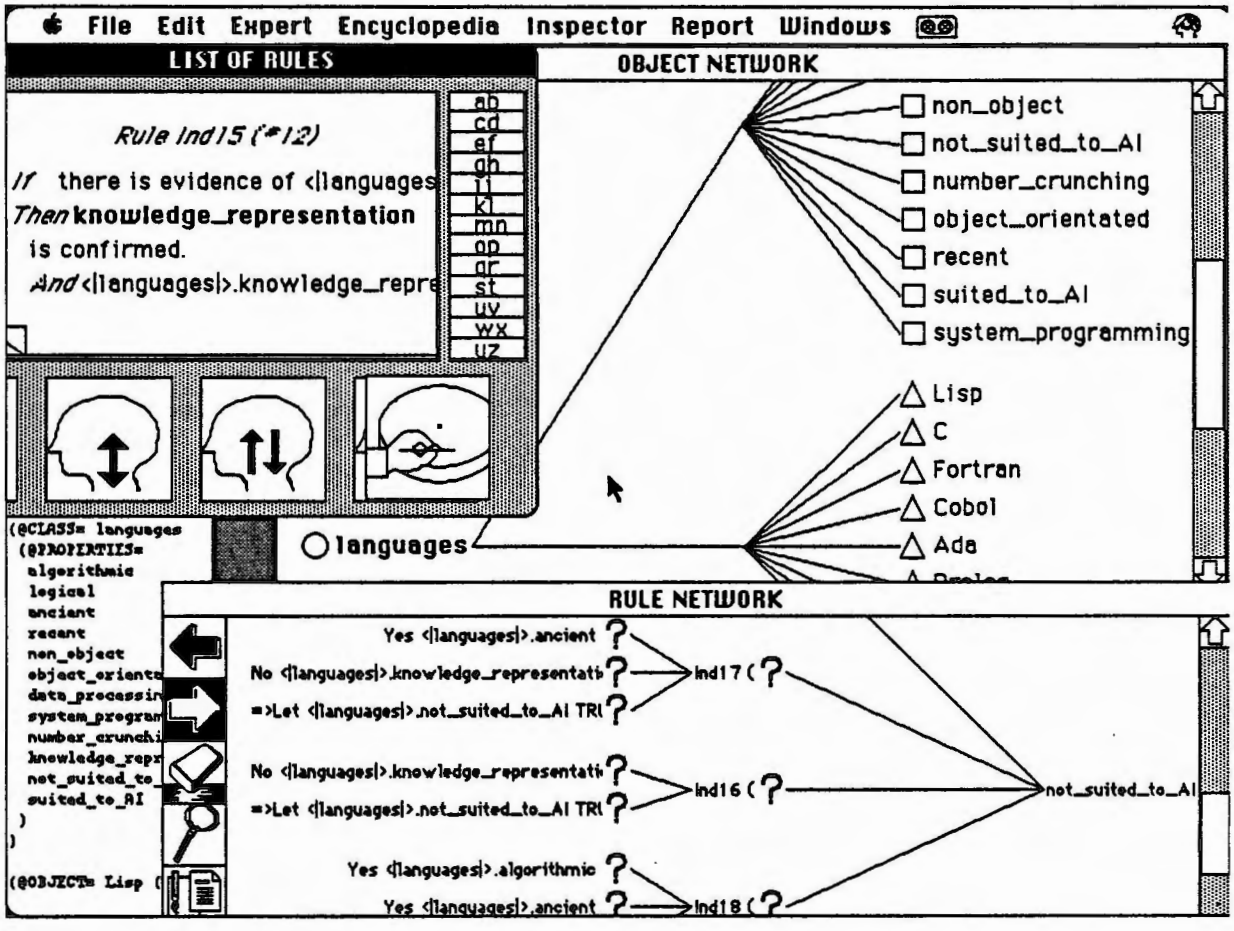


Fig.11 Graphic editing tools after knowledge transfer to NEXPERT

INTERMEDIATE REPRESENTATIONS

In terms of information, the transfer of knowledge into any given formalism is an irreversible task: there is a loss of information due to the limits of the artificial representation. The more steps between the original expression of the knowledge and the performance system which applies this knowledge, the more this effect applies. However, there may be also the emergence of information of a new kind, the derivation of higher-level structures corresponding to the development of the upper levels of the inductive modeling hierarchy (Gaines 1987b). The double effect of loss and gain of information is fundamental in the knowledge acquisition process. It means that while some information may be lost, a key role of the system is also to criticize the knowledge and derive new information. Knowledge acquisition must be as much a deconstructive as a reconstructive process. The technology becomes not only a medium to carry the original knowledge, but also a generative environment to enhance this knowledge.

Decoupling the knowledge acquisition process and the performance system adds risk to the overall process. In return, the pre-processing of the knowledge and its refinement before formalization provide the designer with greater understanding and fewer limitations. Gaining information in the knowledge acquisition phase can be achieved by two means: (1) automated systems that learn and (2) presenting the new information to the human in an understandable fashion. In either case, it is important to ensure that the information gained is understandable and useable.

In the NEXTRA approach, we have incorporated both perspectives. Their association leads to important results for the design of such systems. Since humans perceive the information with their sensory means, the first natural way to make this new information available is to use those means, which translates into graphic, interactive interfaces. For example, in the **Map** tool in NEXTRA: first, the domain expert enters the information using the interviewing scales; then the components analysis is performed by the system and new information is generated, the map of concepts in the space of selected dimensions; then the system creates a bi-dimensional representation of the results which the domain expert can understand and the new information is made accessible.

Figure 12 shows various levels of abstractions for the data structures of NEXTRA and NEXPERT. The initial data structure is a rating along a dimension, a low-level expression of partial knowledge. The output of the **Cluster** tool generates a more global vision which is made very perceptible using the **Map** tool. The generation of rules using algorithms such as Induct generates relations at a level which may be considered a decrease in abstraction. However, when the performance system is running, it provides the user with yet another type of knowledge.

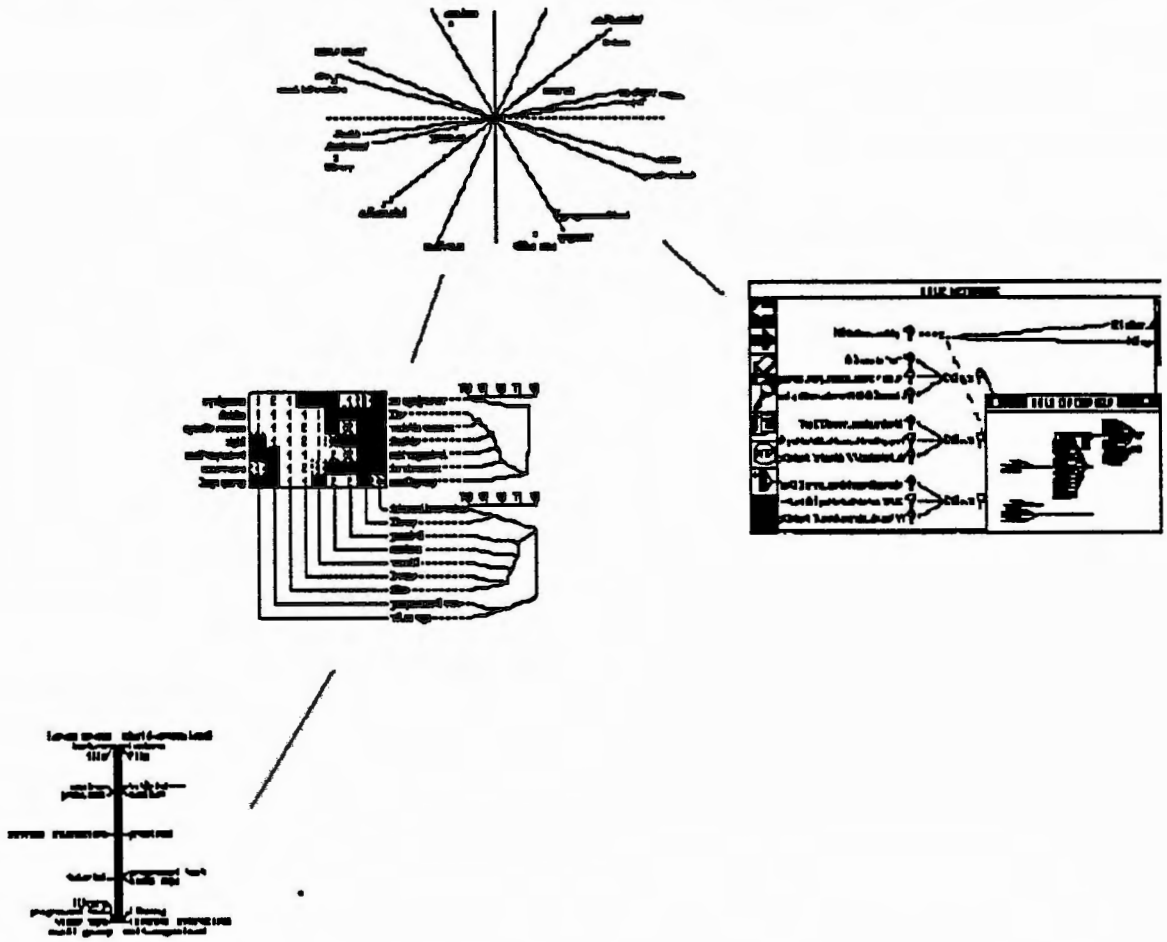


Fig.12 Levels of abstraction in NEXTRA and NEXPERT

TASK DEPENDENCIES

Knowledge formalization and transfer is difficult because many parameters are involved: the nature of the representation, the sources of knowledge, the nature of the task to be modeled and the specific aspects of the domain theory. If, however, the scope of the performance is well-circumscribed to a particular task and that some task-related theory is available, then it becomes easier to generate knowledge for such a system. Amongst the different tasks, some have proven harder to capture than other. Classically, analytic tasks are easier to model while knowledge acquisition for synthetic tasks is more difficult. The simplicity of some classification tasks lends itself to task-level editors and structure or constraint monitoring as well as a well defined debugging cycle. By contrast, a scheduling application typically involves more depth of representation and iterative or non-monotonic behaviors where the control structure is more predominant.

In general, when the reasoning paradigm and representation models are either rather simple or well defined, the knowledge acquisition tools will mainly help reduce the possible mismatches between the actual problem and the performance system. Because of the constraints of the final representation, having intermediate representation might be less useful for whatever is discovered might not be formalizable.

We can now relate the notion of task-dependency to that of intermediate representation. Intermediate representations are less useful when the control mechanisms are fixed. When, the control structure is to be designed, however, an intermediate representation addressing mostly the nature of the chunks of knowledge involved is necessary. Hence, the control structure is built on top of those chunks using either some other knowledge acquisition functions or the computational toolbox of the performance system. In the case of NEXTRA->NEXPERT, the NEXTRA techniques are to be used to mainly elicit the sub-domain conceptual structures, formulating the chunks of knowledge or *knowledge islands* relevant to the domain as shown in Figure 13.

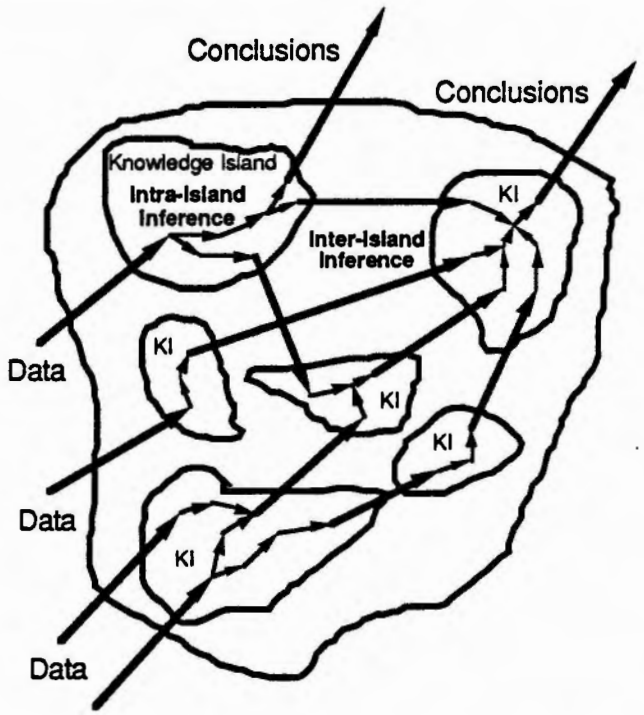


Fig.13 Structuring the domain through knowledge islands

Since the performance system allows one to design a wide variety of tasks, the control structure for transferring the focus of attention between knowledge islands is defined using the performance system's interface . The interface of the performance system itself allows not only to visualize but also to understand the mechanics of the generated knowledge base. From there, the knowledge can undergo multiple refinement and debugging cycles. Thus, in terms of Figure 13, NEXTRA's tools are used within knowledge islands, and NEXPERT's tools are used between them—note that, in these terms, the performance system is also part of the knowledge acquisition environment.

In future developments we intend to achieve even closer integration between acquisition and performance systems, treating the different knowledge representations and different related sub-domains as knowledge bases in their own right. Our overall problem can then be seen as one of cross-deriving, cross-relating and editing these knowledge bases to maintain their mutual consistency. Editing is often regarded as a mundane requirement. However, the variety of related knowledge representations that will usually be present in a knowledge-based system, particularly in the acquisition phase, making the editing of knowledge bases very difficult to support. If representation A, for example text, is processed to give representation B, for example entities, attributes and values, then the editing of the material in representation A should result in changes to that in representation B, and vice versa. However, the relation between representations is usually not functional but merely constrained, that is knowledge in representation A does not determine that in representation B, and vice versa, but only constrains it in some way as shown in Figure 14. Thus, the result of editing knowledge in one representation does not necessarily result in well-defined changes in other representations, only the detection of constraint violations. The editing system should mininimally highlight the violations, but usually it can do more, for example it can adumbrate "likely" changes in other represenations that will remove the violation.

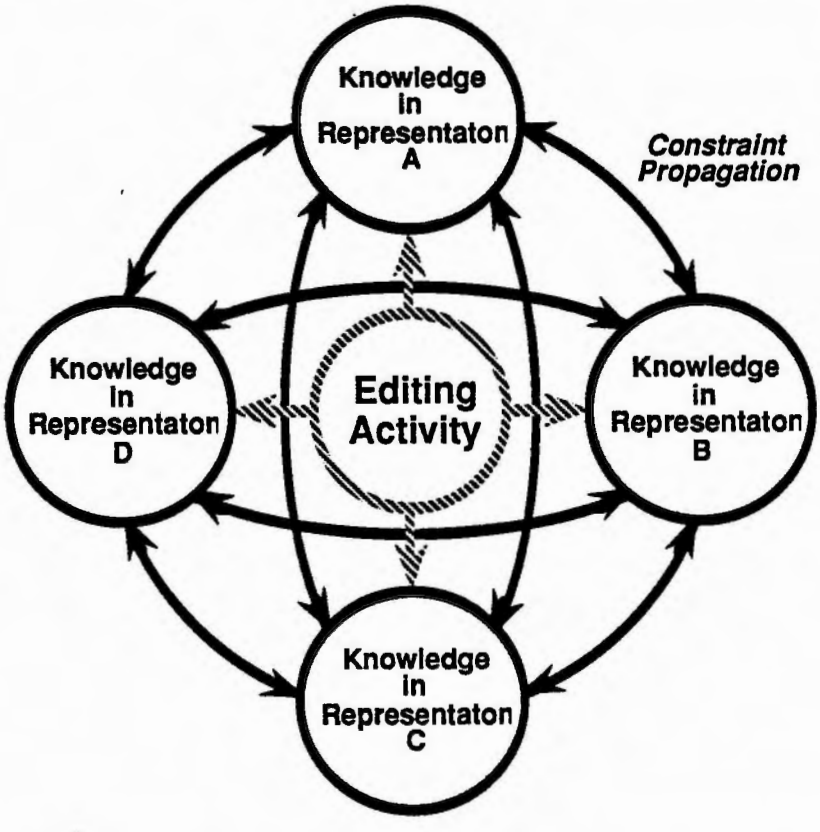


Fig.14 Mutual constraints between knowledge structures in editing

INTERACTIVE TOPOLOGICAL INDUCTION

Induction is a process of knowledge creation based on experience. However, it takes may forms. One can acquire knowledge from a single instance or from a large population of examples, but one also revises this knowledge, based again on any type of experience. One may also simply enrich or refine based on those experiences. In an other case, one compares, one tries and recognize the problem and/or its solution from past experiences. None of these processes are mutually exclusive in human information processing.

Many inductive methods have been described in AI, not including the general literature on the possible theories of induction. In general, the same constraints seem to apply to induction as to problem-solving itself. The main question is: what is the relevant information. As with some AI methodologies which are knowledge-intensive, induction algorithms which generate a sufficient amount of irrelevant knowledge are bound to fail. The mathematical consistency of the methods may provide a framework of validity, but the real framework for validation is whether the generated knowledge is in itself interesting, and not diluted in useless information as well.

Certainly, the algorithms must be mathematically sound, but the mathematical completeness is not necessary. Consider the human operations involved or believed to be involved in the induction process such as similarity, analogy, generalization. They involve the comparisons of abstract structures as well as the definition of operations allowing to transform them or relate them together. In these manipulation is a reflection of the intentions. Inductive algorithms lack the latter.

We believe that two different approaches to the induction problem must be used: (i) providing algorithms which will generate appropriate data structures upon analysis of a data set and (ii) providing an interactive environment where the results of computations are expressed in a graphic form, where the domain expert can discover similarities, differences and relationships.

Figure 15 shows an example of interaction which may provide important information using the **Map** tool. For any given situation (defined as a set of ratings for the constructs along a series of dimensions), there exists one mapping (with specific settings). Any operation such as modifying a rating, as shown on the figure, will yield a different result by principal components analysis and therefore a different map of the concepts. The clusters can be mathematically determined, according to the depth chose in the hierarchical representation (notion of threshold, which can be determined manually or according to some domain knowledge).

Thus, in the mind of the observer we have the following statement: if this event takes place then the overall map (or a subpart of it) is modified in such a way. This formulation, which can naturally be more complex, is then translated into a more domain-dependent syntax and constitutes the body of a new chunk of knowledge. The central argument lies in the power of the graphic representation in eliciting not only the operation but also the ability to formalize its result or impact. At all times, the system makes full use of the perception of the user. In the NEXTRA architecture, we are bringing all aspects of the interaction at the same level of visual abstraction. As a result, the rules will be edited directly from the mapping, but the mapping remains an important intermediate representation for the overall elicitation process.

Another example would be a single level operation whereby the existence and graphic representation of the clusters of concepts elicits the construction of relations between them. This is illustrated in the following Figure 16.

The **Cluster** and **Map** tools are based on mathematical distance-based analyses, but their output is an image as opposed to expert system code. They require another level of interactive analysis before generating the code in question.

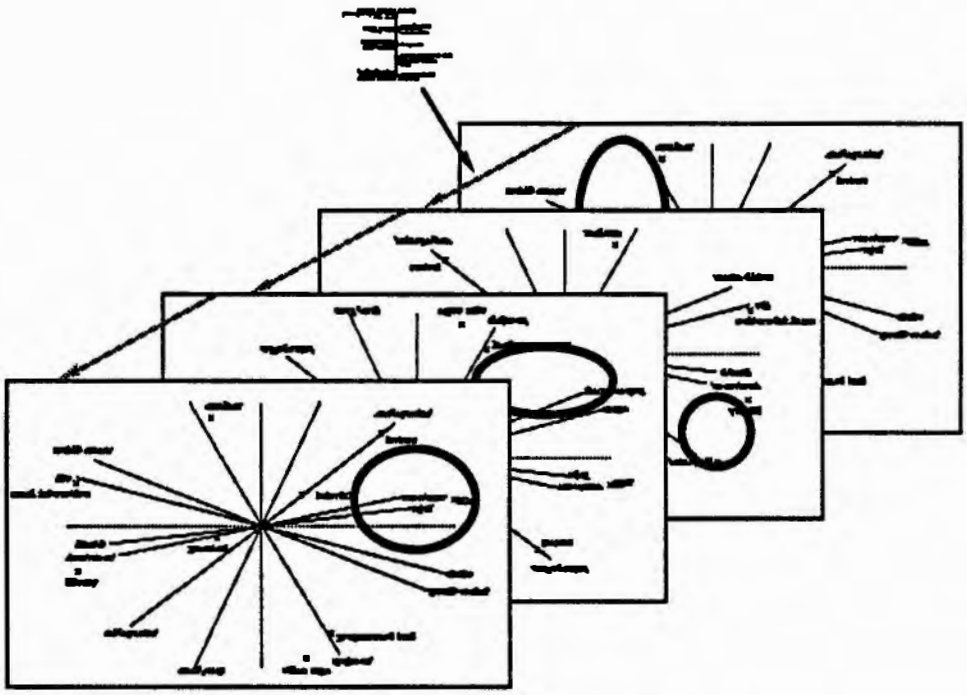


Fig.15 Knowledge analysis and creation in the graphic domain

By introducing the developer in the inductive process which makes the knowledge acquisition incremental, we reduce the amount of irrelevant information which may be generated by pure inductive techniques. What we may introduce though is "visual noise", or noise coming from the inherent lack of completeness of the human processes. We believe, however, that this is a more powerful approach, driven by intention and relations. Brittleness of the overall system will also come from the generated rules and the validity of the generalizations made by the system or elicited from the expert. Again, different means can be used. The more computational approach will try to keep degrees of certainty or decreases in information underlying any assumption. The latter should then be reflected in the rules and taken into account either by the inference engine or by some other technique interacting with the engine. At another level, closer to the graphic approach to the bounded rationality, we envision direct access to hypertext support, from either the concepts mapped in NEXTRA, or as is the case currently, at the level of the data represented in the rules.

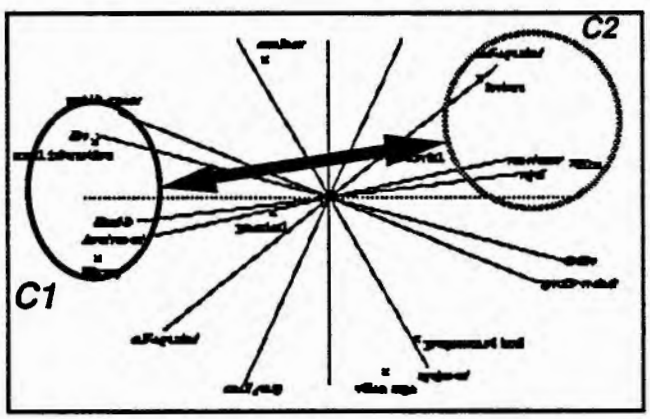


Fig.16 Relations in the graphic domain

FROM STATES TO TRAJECTORIES

Each operation performed in NEXTRA such as the modification of the rating of an attribute can be represented as a modification of the **Map** generated graph. As suggested above, the comparison between the two states may elicit a particular chunk of knowledge from the domain expert. Mathematical methods can be designed which will also formalize this transformation. However, comparing final and initial states can be a tedious approach if the final state is not well known in advance, or is too different from the initial state. When for instance many variables are involved which evolution follows different functions, it becomes necessary to be able to draw the trajectory followed by a particular entity or cluster. In order to obtain such a representation, the operations performed by the inductive tools must be made highly incremental. In order to achieve this, analytic models of the transformations are required. We are currently working in this direction.

Figure 17 illustrates the relative positioning of an entity in terms of the cluster to which it belongs, as a function of the evolution of one of its attributes A1. The initial state is described on top of the diagram, and the final state at the bottom. The final state is reached earlier than this (intermediate square). The shape of the curve may thus be indicative of the "fragility" of the initial description. In other terms, it is possible to observe the stability of the system's state with respect to one or more variables. The objective is to represent the nature of the state of the system and its evolution in a graphic form so as to elicit abstract knowledge, which would otherwise most likely remain buried in the expert mind. If such techniques allow the uncovering of expert knowledge, they certainly provoke the reflection on whatever knowledge is available and may also lead to the discovery of new knowledge.

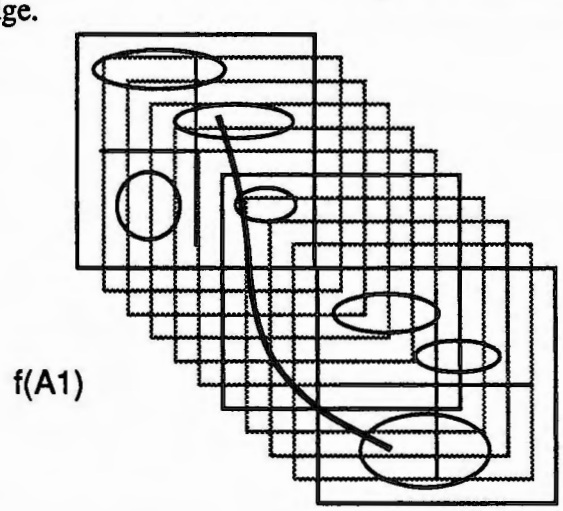


Fig.17 Trajectory of evolving knowledge in the graphic domain

INTERACTION, VISUALIZATION AND VALIDATION

We have emphasized in this paper the significance of user interaction with knowledge-based systems and the provision of facilities for users to visualize the knowledge base from different perspectives. This emphasis recognizes the role of knowledge-based technologies as supporting knowledge processes in human society, substituting for them only when appropriate, and even then giving easy access to the process if required. Hawkin's (1983) studies of the behavior of experts and their clients, and the management literature on professional development (Schön 1983), suggest that expert-client interaction is a process of negotiation, modeling, and continuing acquisition, rather than authoritarian decision-making based on a static knowledge-base. It is this dynamic process of knowledge exploration which is our target for an integrated system.

This emphasis on visualization and exploration is not in conflict with the use of the system to make 'quick and dirty' decisions with no exploration and no explanation. Quite the contrary—it is the potential for full exploratory visualization which allows users to trust in more routine applications of the system, knowing that a full audit trail of all the knowledge processes is always open to them. In particular, meaningful visualization is a powerful technique for highlighting situations with which the system is not designed to cope. Such anomaly detection is important not only in making available to clients meta-information about the suitability of their problem for solution by the system, but also in closing the inductive knowledge acquisition loop back to the expert enabling him or her to examine novel situations that might otherwise go undetected (Gaines 1988).

The central core about which our thinking revolves may best be characterized as one of *validation.* Not just the validation of the performance system in the quality of its ultimate operation, but the ongoing validation of the knowledge base at all stages and in all forms. The internal cross-consistency checks of Figure 14 are a form of validation. The visual relationships in Figures 5 through 7, 11, and 15 through 17, all support direct human validation of the knowledge base as simply and naturally as possible. Our requirement to be able to move the knowledge base back and forth freely between the acquisition and performance tools is precisely to support a continuous knowledge validation process. Knowledge base management requires the careful methodologies developed for data base management supplemented by techniques that recognize and support the volatility, fallibility, vagueness and inconsistencies of knowledge.

Figure 18 shows what we are trying to achieve at the next stage in the forms and relations of knowledge that will be supported in the combined tool. All levels and forms of knowledge exist in some form in NEXTRA and NEXPERT systems currently. The task is to achieve as seamless an integration as possible with the only discontinuities being those intrinsic to knowledge, not those introduced by the tools.

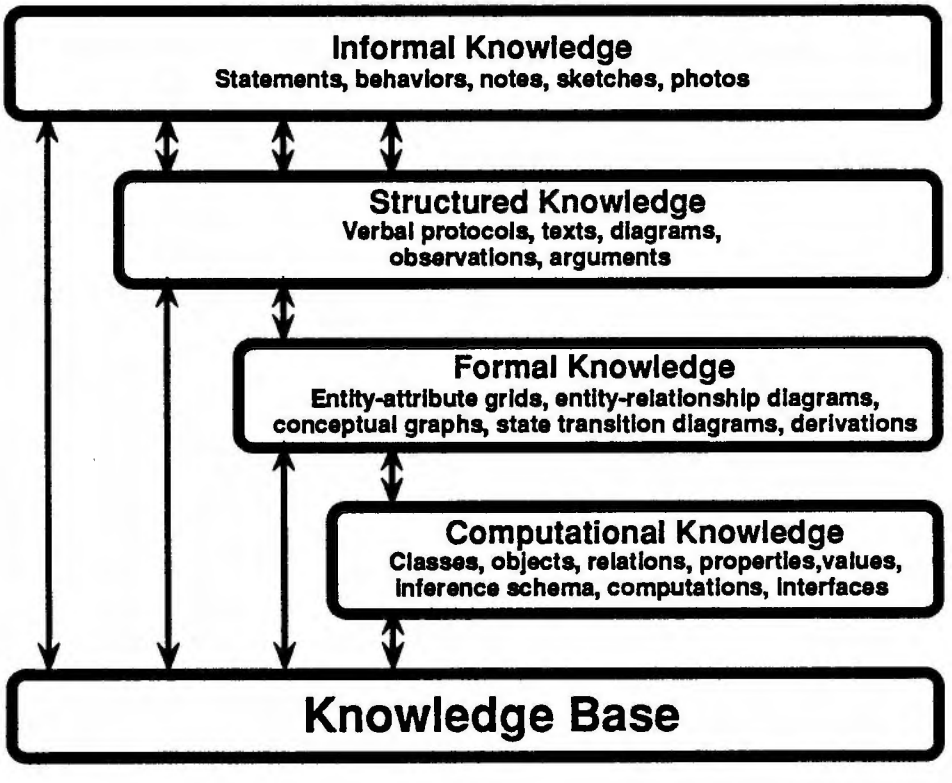


Fig.17 Trajectory of evolving knowledge in the graphic domain

CONCLUSIONS—TOWARDS A KNOWLEDGE SUPPORT TOOLBOX

We see the integration the techniques described in this paper not only as powerful new tools for the acquisition of knowledge but also as the initial approach to building of a toolbox of technologies including relational and other databases, hypertext, knowledge structuring tools, CD-I based and on-line information systems, all managed by the AI performance system, and incorporating integral knowledge acquisition.

This integration of technologies is both a natural direction of development in knowledge-based systems and a necessary step towards supporting the full range of knowledge processes in human society.

Neither the computational nor the cognitive foundations of knowledge-based systems are sufficiently advanced for it to be possible to give strong normative guidelines for the design of the type of integrated system discussed in this paper. The only approach possible is an empirical one, founded on the available knowledge in AI, guided by intuition, and leading to the production of tools that can be tested in real-world applications. NEXTRA and NEXPERT *OBJECT* are now being used by a very diverse community, and it is the experience of these users that will determine whether we are succeeding in providing an effective toolbox supporting the development and application of knowledge-based systems.

ACKNOWLEDGEMENTS

We are grateful to many colleagues for discussions, particularly at the Knowledge Acquisition Workshops, that have influenced this paper. In particular we would like to thank John Boose, Jeff Bradshaw, and Mildred Shaw, for access to their own research and many stimulating discussions.

REFERENCES

Boose, J.H. (1986). **Expertise Transfer for Expert System Design.** 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** (1), 3-28 (January).

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Gaines, B.R. (1987a). Rapid prototyping for expert systems. Oliff, M.D., Ed. **Intelligent Manufacturing: Proceedings from First International Conference on Expert Systems and the Leading Edge in Production Planning and Control.** pp.45-73. Menlo Park, California, Benjamin Cummins.

Gaines, B.R. (1987b) An overview of knowledge acquisition and transfer. **International Journal of Man-Machine Studies 26**(4), 453-472 (April).

Gaines, B.R. (1988). Positive feedback processes underlying the formation of expertise. **IEEE Transactions on Systems, Man & Cybernetics,** to appear.

Gaines, B.R. & Shaw, M.L.G. (1986). Induction of inference rules for expert systems. **Fuzzy Sets and Systems,** 8(3), 315-328 (April).

Hawkins, D. (1983). An analysis of expert thinking. **International Journal of Man-Machine Studies, 18**(1), 1-47 (January).

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

Quinlan, J.R. (1987) Simplifying decision trees. **International Journal of Man-Machine Studies 27** (3), 221-234 (September).

Rappaport, A. (1987a) Multiple-problem subspaces in the knowledge design process. **International Journal of Man-Machine Studies 26**(4), 435-452 (April).

Rappaport, A. (1987b) Cognitive primitives. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.15-0-15-13. Banff (October).

Schön, D.A. (1983). **The Reflective Practitioner.** New York: Basic Books.

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. (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. (1986). Interactive elicitation of knowledge from experts. **Future Computing Systems, 1**(2), 151-190.

Shaw, M.L.G. & Gaines, B.R. (1987a). An interactive knowledge elicitation technique using personal construct technology. Kidd, A., Ed. **Knowledge Eliciation for Expert Systems: A Practical Handbook.** pp.109-136. Plenum Press.

Shaw, M.L.G. & Gaines, B.R. (1987b) KITTEN: Knowledge Initiation & Transfer Tools for Experts & Novices. **International Journal of Man-Machine Studies 27**(3), 251-280 (September).

Shaw, M.L.G. & Gaines, B.R. (1988). A methodology for recognizing consensus, correspondence, conflict and contrast in a knowledge acquisition system. **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff (November).

Shriver, B. & Wegner, P. (1987). **Research Directions in Object-Oriented Programming.** Cambridge, Masachusetts: MIT Press.

Slater, P., Ed. (1976). **Dimensions of Intrapersonal Space: Volume 1.** London: John Wiley.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

The Meaning Triangle as a Tool for the Acquisition of Abstract, Conceptual Knowledge

Stephen Regoczei

Computer Studies
Trent University
Peterborough, Ontario
CANADA K9J 7B8

Graeme Hirst

Department of Computer Science
University of Toronto
Toronto, Ontario
CANADA M5S 1A4

ABSTRACT

The meaning triangle is presented as a useful diagramming tool for organizing knowledge in the informant-analyst interaction–based, natural language–mediated knowledge acquisition process.

In concepts-oriented knowledge acquisition, the knowledge explication phase dominates. During the conceptual analysis process, it is helpful to separate verbal, conceptual, and referent entities. Diagramming these entities on an agent-centered meaning triangle clarifies for both informant and analyst the ontological structure that underlies the discourse and the creation of domains of discourses.

1 INTRODUCTION

In the construction of knowledge-based systems, even in cases where the domain of application is fairly concrete and straightforward, a considerable amount of abstract, conceptual knowledge must be acquired. This abstract knowledge is often hidden, tacit, commonsense, and "too obvious to see"; hence informants and analysts have difficulty describing it. Often careful distinctions have to be made between verbal, conceptual, and external entities to bring this knowledge to light. Extensions of the meaning triangle, introduced in this paper, are useful diagramming tools to accomplish this task.

In looking at knowledge acquisition as a *process*, Regoczei and Hirst (1988) divided it into three stages (Figure 1): elicitation, explication, and formalization. *Conceptual analysis* is the main technique in the explication of *any* knowledge, but it plays a special role when the knowledge is

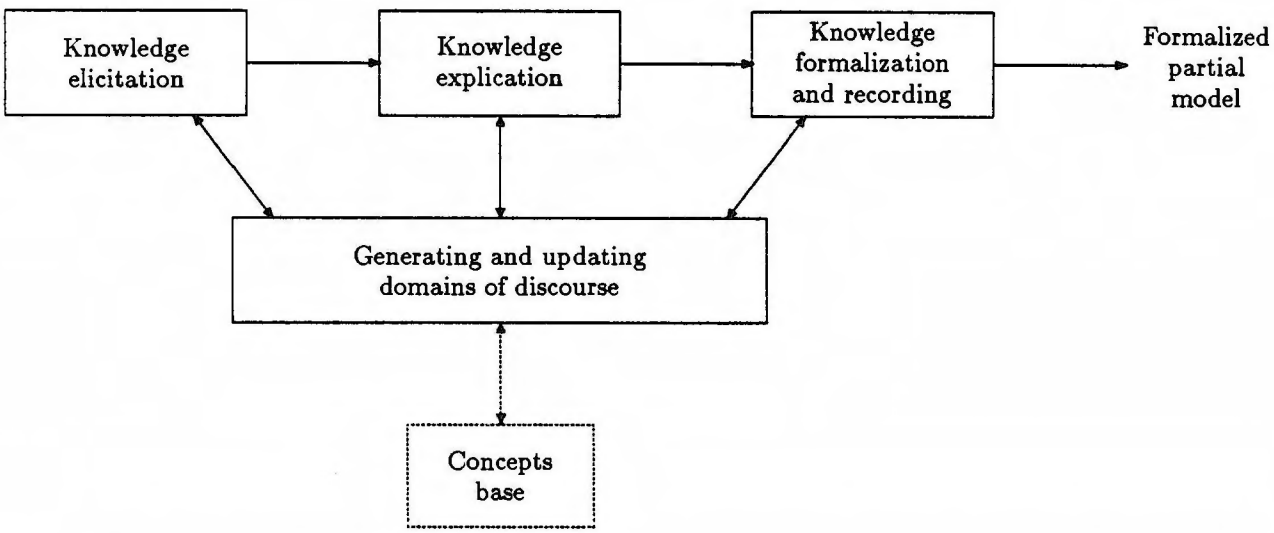


Figure 1: Processes in knowledge acquisition. The input to all processes is the mental models of the analyst and the informant. The output is a formalized *partial* model of domains of discourse.

not about physical objects but rather about abstract entities, which are social constructs. The objective of the knowledge acquisition process is to create and update a *domain of discourse* that contains the knowledge. This domain of discourse is to be described, in a machine-usable form, in a notation such as conceptual graphs (Sowa, 1984; Regoczei and Plantinga, 1987; Regoczei and Hirst, 1988).

Knowledge acquisition can be viewed as *interaction*. We restrict our attention to the case where the domain expert, the informant, is interviewed by an analyst. The discussion between informant and analyst produces tape-recorded and transcribed text. This natural language text is the raw material from which the analyst reconstructs the domain knowledge and builds a formal, partial model of those portions of the expertise that are required for the construction of the knowledge-based system.

The informant-analyst–based, natural language–mediated knowledge acquisition process can also be regarded as a process of *mutual attribution*. Adopting the *mental models hypothesis* (Sowa, 1984; Regoczei and Plantinga, 1987; Regoczei and Hirst, 1988), *i.e.*, that people understand the world by forming mental models, we can picture the task of the analyst as building mental models of the informant's mental models. More precisely, the analyst re-creates the informant's knowledge by building mental models that the analyst then "attributes" to the informant, and labels as the model of the mental models of the informant. The correctness of the labeling, and the verisimilitude of the mental models so labeled, is tested by the analyst, who expresses his views, both in terms of colloquial natural language and in more formal terms such as logic and diagrams, and asks for feedback from the informant. Corrective measures are taken as required. In effect, the informant and analyst negotiate what counts as (relevant) knowledge. The roles of the informant and analyst are symmetric. Each is building mental models of the other's mental

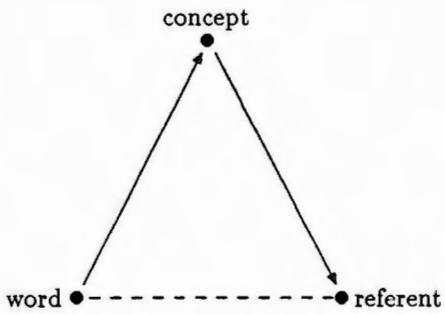


Figure 2: The meaning triangle.

models, and *harmonizing* them through interaction mediated by natural language.

In brief, the analyst does not merely "acquire" something that already exists. Rather, during the knowledge acquisition process, the informant and the analyst cooperatively create a domain of discourse—or perhaps several *distinct* domains—by matching concepts to words.

2 THE MEANING TRIANGLE

The meaning triangle (Figure 2) was prominently featured by Ogden and Richards in their book *The Meaning of Meaning* (1923). It has also been used by Sowa (1984) to motivate his conceptual graphs notation, by G.D. Martin (1975) for the analysis of literary texts, and by Umberto Eco (1976) for semiotic analysis. It was introduced in knowledge acquisition by Regoczei and Hirst (1988).

The main function of the meaning triangle in knowledge acquisition is to help draw distinctions between *words, concepts*, and *referents*.

- Words, or lexical objects in general, are concrete linguistic entities embodied in speech or in written text. They are pictured as located on the word node of the meaning triangle.
- Concepts are abstract, intangible entities, usually associated with human thinking. There is considerable controversy surrounding the existence and nature of concepts. In this paper, as well as in knowledge acquisition work in general, one takes a rather pragmatic approach to concepts, considering them to be constructs that are used to build mental models of the world. They are tools to aid thinking. Concepts as cognitive constructs are pictured as being located on the concept node of the meaning triangle.
- The referent node is the place for entities such as physical objects that are pictured as being part of the external world.

3 ABSTRACT, CONCEPTUAL KNOWLEDGE

The referent node seems simple, but is actually highly problematic. The entities that populate the referent node are usually construed as "objects out there"—hence physical objects are the prototypical example. These are "objects out there" because they are independent of what any person thinks or says. The qualifier "out there" is to capture the intuition that words can be uttered, and thought can be generated, but no amount of thinking or talking is going to bring a table or cat or chair into being.

While this intuition is very strong and clear, and exercises a powerful influence upon our thinking, it is not too difficult to blur the edges. There are socially constructed abstract entities, especially in the fields of law, culture, interpersonal interactions, organizations, and mathematical sciences, which *are* brought about through talking and thinking by individuals within a group. To a single individual, these abstractions may appear as given as the earth and the sky. Yet they are created, and hence may be changed, by people acting together. But the process is not wholly arbitrary; it takes place within the constraints and limitations of the physical and social world. The "software" of such constructs often requires a "hardware" component. For example, armies and corporations are abstract entities with heavy "hardware" components. Although they are abstract, they are much more bound by the physical world, and are much more concrete, than, let us say, the constructions of mathematics. They can be thought of as ***hybrid entities***, with lexical, conceptual, and physical components.

In brief, if private, cognitive concepts are to be diagrammed on the meaning triangle by placing them on the concept node, then public, abstract concepts may be placed, with good justification, on the referent node. Acquiring knowledge about these abstract entities, which are essentially social constructs, is the challenging task that we are addressing in this paper. The task is made harder by noting that information or clues about these abstract entities is available mostly through the mental models of individuals and what these individuals utter about their mental models. We may call this difficulty the ***perspectivist paradox***.

4 AN EXAMPLE: "FIVE CHAIRS"

Esoteric knowledge may have its own intrinsic fascination, but for the building of expert systems, knowledge-based software, or conceptual schemas for databases, the required knowledge is usually quite mundane in nature. Yet even in these simple cases of knowledge acquisition, conceptual knowledge may play a central role.

To take a very basic and simple example, let us assume that we are building a domestic robot with housekeeping expertise. When we interview a resident of the house the robot is to clean, he may utter a statement such as this:

(S1) "There are five chairs in the room."

There are two different types of knowledge being alluded to in this statement. The first type may

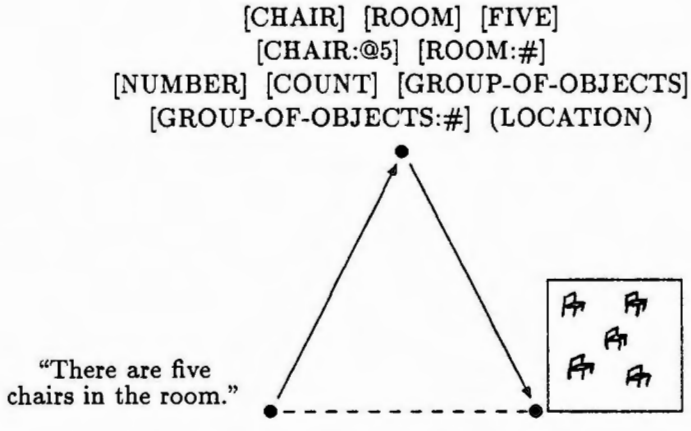


Figure 3: Meaning triangle for statement S1.

be characterized as inventory-like (Regoczei and Plantinga, 1987; Regoczei and Hirst, 1988). The claim is made that there are five chairs in the room, not four or eight. This kind of propositional, declarative knowledge is typically what is acquired in knowledge acquisition that is oriented towards rules, facts, and propositions. Such knowledge is often acquired only at the verbal level using what is nothing more than uninterpreted character strings.

But there is an even more important kind of knowledge being alluded to by statement S1, and this is pertaining to what exists—ontological knowledge (Regoczei and Plantinga, 1987; Regoczei and Hirst, 1988). What is being said, by convention, generates the presupposition, or the claim, that certain things exist. The informant is talking about chairs, and a room, and the concept of containment. He is talking about numbers, in particular the number five, the ability to count objects, and the notion of grouping objects together to be counted.

This gives us a large number of concepts, which we can list using Sowa's conceptual graphs notation for concepts, and we can diagram the proposition "behind" the statement as follows:

[CHAIR:@5] → (LOCATION) → [ROOM:#].

In diagramming this analysis on a meaning triangle (Figure 3), we place statement S1 on the word node, and the various concepts contained in the ontology are located on the concept node. If there is an actual, physical room that is being talked about, and there are actual physical chairs in it, then what is diagrammed on the referent node is the information that pertains to the state of the object domain "out there". The domain of discourse on the concept node should contain not only the general or generic concepts [CHAIR] and [ROOM], but also the specific concepts that refer directly to the particular state-of-affairs within the object domain, *i.e.*, the action defined by the concept [COUNT], as applied to a particular [GROUP-OF-OBJECTS:#], yielding [CHAIR:@5] in a particular room specified by the concept [ROOM:#].

Item	Notation
word	"fish"
generic concept	[FISH]
concept, pointing to a referent	[FISH:#]
referent	Informal English description, diagram, or a specific concept name such as [FISH:#2356] as a surrogate

Figure 4: Summary of conceptual graph notation.

5 A MOLECULAR THEORY OF KNOWLEDGE

We have been diagramming knowledge on the concept node not as incoherent, amorphous clouds, but as hard-edged, individual, discrete concepts. This practice needs some explanation, since when people think and talk naturally, they do not "step" carefully from one thought to the next. Their thoughts have a tendency to flow into each other, and be forever changing, like an anthill (Regoczei and Plantinga, 1987).

In the analysis above, we tacitly assumed that knowledge can be "chopped up" into units that are conceptual in nature, that these units can be named in appropriately mnemonic ways, and that these units, recombined into conceptual structures, capture the knowledge required. This assumption can be described as the Molecular Theory of Knowledge.

We certainly do not assume that all knowledge is molecular in nature. On the contrary, much of cognitive activity seems to be sub-conceptual. Nevertheless, we state that in order to build software, we have to find conceptual models for expertise and world knowledge, because, given the state of today's technology, only conceptual models of knowledge are tractable computationally. In discretizing knowledge, we make it easier to handle the mapping between text and concepts, while at the same time avoiding certain controversial theories that claim that all thinking and knowledge is language-based.

The representation of knowledge could be effected in any of several different notations. One would prefer those that emphasize concepts as the basic semantic primitives, rather than logical connectives, predicates, or functions. For this reason, we adopted the notation of Sowa's conceptual graphs (1984). Some of this notation we have already used informally. We summarize the notational conventions of conceptual graphs in Figure 4.

6 EXTENSIONS OF THE MEANING TRIANGLE

While it is intuitively suggestive, the traditional meaning triangle does not stand up to close scrutiny. The issues that have to be handled in real-life knowledge acquisition interactions between analyst and informant force the extension of the traditional meaning triangle.

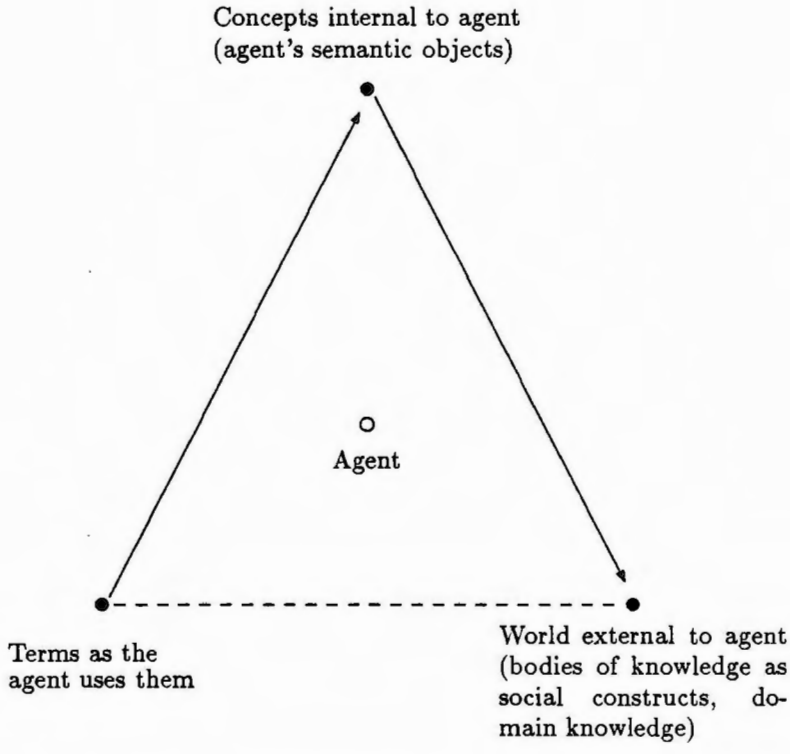


Figure 5: An agent-centered meaning triangle.

6.1 The agent-centered meaning triangle

In its traditional form, the meaning triangle provides no guidance as to where to place a public concept. Much of the hesitation in using the meaning triangle for purposes other than literary text analysis came from this shortcoming. The problem, already indicated above, is this: because it is a concept, an abstract public concept should be placed on the concept node. But because a public concept is also a social construct "out there", it should be placed on the referent node. To resolve this deadlock, we introduce the *agent-centered meaning triangle*. The agent-centered meaning triangle has an extra node for specifying the agent relative to whom a given entity is to be classified as verbal, conceptual, or referent in nature (see Figure 5).

6.2 The meaning trapezium

The concept node can be split to accommodate the distinction between forming a concept on the basis of hearing some words, and forming a concept on the basis of observing something in the object domain (Martin, 1975). Splitting the concept node in two (Figure 6) creates the meaning

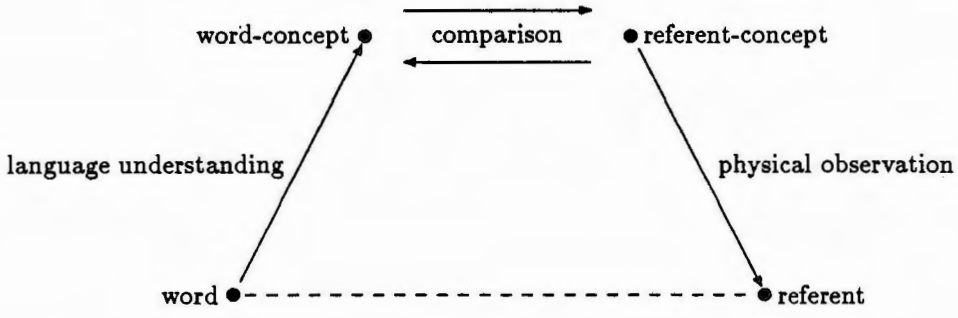


Figure 6: The meaning trapezium.

Meaning triangle:

Name or description: [*name*]
Type: [*e.g., agent-centered*]

Word-node: [*a description of the entities that populate the word node*]

Concept-node: [*a description of the entities that populate the concept node*]

Referent-node: [*a description of the entities that populate the referent node*]

Agent-node: [*a description of the agent, if applicable*]

Figure 7: Frame-like notation for the meaning triangle.

trapezium. All comparisons between a generic concept such as [CAT] and a specific concept such as [CAT:#] or [CAT:#235] take place at the conceptual level. Although we do not claim that there is a sortal distinction between these concepts, drawing such distinctions does enable us to make precise the difference between "speaking in general" and "talking about a specific thing".

7 A FRAME-LIKE NOTATION FOR THE MEANING TRIANGLE

Rather than drawing pictures, one can define a frame-like description for meaning triangles. Figure 7 shows a schema for the notation. This notation is especially useful for producing modifications and alternative versions of the same meaning triangle. Changes may be noted without repeating the full information.

Meaning triangle:

Name or description: Meaning triangle for a mathematician trying to talk about the Riemann integral.

Type: Agent-centered.

Word-node: Statements to analyst.

Concept-node: The mathematician's private mental models under the concept label [RIEMANN-INTEGRAL].

Referent-node: The Riemann integral, a public, abstract construct, created as a social construct by the community of mathematicians.

Agent-node: Mathematician acting as the informant.

Figure 8: Agent-centered meaning triangle for mathematician.

8 AN EXAMPLE: THE RIEMANN INTEGRAL

Abstract, conceptual knowledge is encountered both at the commonsense, everyday level as discussed above, and at the theoretical level in working with abstract theories. We now turn our attention to abstract, public bodies of knowledge, concentrating on the perspectivist paradox stated above, *i.e.*, that information or clues about public concepts are available mostly through the private mental models of individuals.

Mathematics provides the richest area for examples in the acquisition of knowledge about abstract, conceptual entities. The conceptual analysis required to explicate this knowledge is usually difficult. Ontological issues dominate the informant-analyst interaction. One has to be able to separate mathematical "reality" out there, usually construed as "platonic entities" or "mathematical truth" out there, from what individuals think about these mathematical entities and theories. Mathematicians are careful to avoid psychologism or mentalism, by objectifying "mathematical truth", picturing it as "eternal", "immutable", and independent of the opinions of mere mortals. Nevertheless, the knowledge acquisition analyst is dealing with a real, live informant, and has nothing more to go by than what this informant thinks and says. These issues are central to the acquisition of abstract, conceptual knowledge in general, and the acquisition of mathematical knowledge is a good test case to study, but, not surprisingly, these issues are seldom discussable within the discourse of mathematics. That is why we so often find people complaining that mathematicians do not know how to explain things.

We shall use the example of an analyst trying to find out about the Riemann integral from a mathematician. On an agent-centered meaning triangle, we can diagram the discourse with the frame-like representation, as shown in Figure 8.

The [RIEMANN-INTEGRAL] is the agent's mental model—possibly consisting of a large, structured *cluster* of concepts—of the public concept of the Riemann integral. In philosophical disputes,

Meaning triangle:

Name or description: Meaning triangle for a mathematician trying to talk about the Riemann integral; Version 2.

. . . .

Concept-node:

Domain of discourse number 1:

Private, containing the concept [RIEMANN-INTEGRAL].

Domain of discourse number 2:

Private, labeled as the "surrogate" domain, containing mental models of the public entity on the referent node.

. . . .

Figure 9: Updated meaning triangle for mathematician.

the existence of the public concept of the Riemann integral may be called into question. Since in this paper we assume the position of the "reflective practitioner" (Schon, 1983), we have no hesitation in declaring that mathematical knowledge (like medical knowledge, legal knowledge, and so on) exists as a social construct independent of the idiosyncrasies of individuals (but not as a platonic entity). This is the public knowledge that is located on the referent node. The private concept of [RIEMANN-INTEGRAL], however, is relativized to the agent.

During the knowledge elicitation and explication stage, the agent may be uttering statements either about the mathematical knowledge "out there" or about his own personal view, or picture, of this mathematical knowledge. This means that a further distinction will have to be drawn. Some statements may be about [RIEMANN-INTEGRAL] on the concept node, and some about the public, abstract concept of the Riemann integral on the referent node.

This generates a paradox. The public concept of the Riemann integral is accessible to the agent only insofar as he is able to form a mental model of this public concept. So, actually, there may be two domains of discourse, simultaneously located on the concept node, one identified as the "surrogate" domain which "faithfully" reflects the public concept, and the other which corresponds to the more private opinions held by the mathematician. We can update the meaning triangle of Figure 8 as shown in Figure 9.

There is often no effort on the part of the informant to keep separate his private domain of discourse from what we could label as his "surrogate" domain. In fact, he may often deliberately blur the distinction, if he is aware of it at all, in order to get his own personal views on record. Similar blurring may occur in cases when the expert's own way of looking at things differs from the "official" way of doing something. The process may work in reverse in the case of medical diagnosis, where the doctor will quote the medical textbook rather than reveal his own "informal" but quite effective techniques.

Separating and clearly labeling the knowledge about the private domain of discourse and the knowledge about the "surrogate" domain is the responsibility of the analyst. What the informant says, in his own words, usually contains enough clues for the analyst to capture the required distinctions. For example, the mathematician might say:

> Well, let me tell you how I think about the Riemann integral. You won't find this in books. The books describe things differently. It always bothered me that I couldn't immediately see the difference between the Lebesgue integral and the Riemann integral ...

From this text it is clear that we need two domains of discourse, the first containing the mathematician's private views, and the second his accurate reflection of the "official" story. In the acquisition of expertise, both domains are of interest.

The informant may continue describing his mental models by saying, for example:

> For the Riemann integral, the partitions on the domain are too crude, it's like a giant with big fingers not being able to pick up the small pieces, or a net with thick strands and large holes that lets the plankton and small fish slip through as if they were not there.
>
> Point sets with non-zero measure just slip through the fingers ... just slip through the net in little bits and pieces .... I also think of the ordinate set as the whale's teeth, straining the plankton, getting something out of a lot of little nothings.

This is not exactly conventional, mathematical prose. In fact, a mathematician may feel rather uneasy talking this way in public. Although the imagery is concrete, the essential concepts "behind" the imagery are abstract. Concepts such as [POINT-SET], [MEASURE], [NON-ZERO-MEASURE], and [ORDINATE-SET] are standard mathematical concepts, but the [RESOLUTION-POWER] concept that the informant implicitly alludes to is not. To capture the informant's private domain of discourse, in addition to the "official" story, one has to accept concepts that have no direct referent in public, abstract, mathematical knowledge.

9 A MEANING TRIANGLE OF DOMAINS

Perhaps the most important extension of the meaning triangle, as already indicated above, is formed by placing not only single entities, but entire domains on the vertices. An agent-centered version of such a meaning triangle is shown in Figure 10. The domains are not necessarily formed by the agent himself. The domains may be abstract entities created by an observer who watches the process and diagrams it in the form of a meaning triangle. Thus the various domains—domains of discourse, "surrogate" domain, object domain—may all be creations at a higher level of abstraction. It is at this level that one may draw taxonomic distinctions between processes, events, qualities, attributes, and so on, which are crucial for a smooth flow of the interaction between analyst and informant (Regoczei and Hirst, 1988). At times, the expert seems to be

adept at manipulating inconsistent and conflicting knowledge. The contradictions implicit in his statements often can be resolved by drawing distinctions between concepts, and through the creation of separate domains of discourse. For an especially vivid example to illustrate this point, we will use Eddington's paradoxical story of his "two tables" below.

In Figure 10, the various domains of discourse are diagrammed on the concept node. These concepts and conceptual structures are the constructs of the agent. These domains of discourse may contain concepts of possible events, processes, and so on. One or more domains carry special labels as "surrogate" domains reflecting "reality" as it "really" exists "out there".

The referent node contains the object domain that includes both physical objects and abstract entities as social constructs. One may include events and processes that take place in the object domain. What characterizes these is that they unfold in real time. In fact, the referent node is dominated not only by physical space, but also by real physical time.

The population of the word node provides interesting insight. Essentially, there are two classes of statements:

- Descriptions of domains of discourse, using text and diagrams.
- Descriptions of the object domain based upon certain specific perspectives given by the "surrogate" domain.

In ordinary, everyday use of language, we do not distinguish whether we are talking about the population of the concept node or the population of the referent node. Part of the power of natural language derives from this systematic disregard of typing rules through the use of *deliberate category mistakes*. In certain cases, however, during the knowledge acquisition process, one may want to specify, using the distinction on the word node, whether one is talking about concepts in general or concepts directly interpretable within the object domain.

10 USES OF THE MEANING TRIANGLE

As discussed above, the drawing of distinctions between categories of existence such as lexical, conceptual, external referent-like, and agent-like is crucial in the informant-analyst interaction. Extensions of the meaning triangle, introduced in this paper, are helpful in documenting these distinctions. "What are we talking about?" is the basic question that underlies the discussion of the analyst and informant as they are trying to understand each other. They have to draw distinctions to sort out contradictions and terminological confusion. They have to handle misunderstandings created by semantically ill-formed dialogue, create new conceptual entities as needed, effect appropriate *concept cluster attachments*, and harmonize their mental models.

Detailed examples of the use of the meaning triangle in systems analysis and requirements specification are discussed in "Knowledge acquisition and requirements specifications: Uses of the meaning triangle" (Regoczei and Hirst, forthcoming). In this and the next section, we briefly outline some examples from this paper to illustrate the key issues.

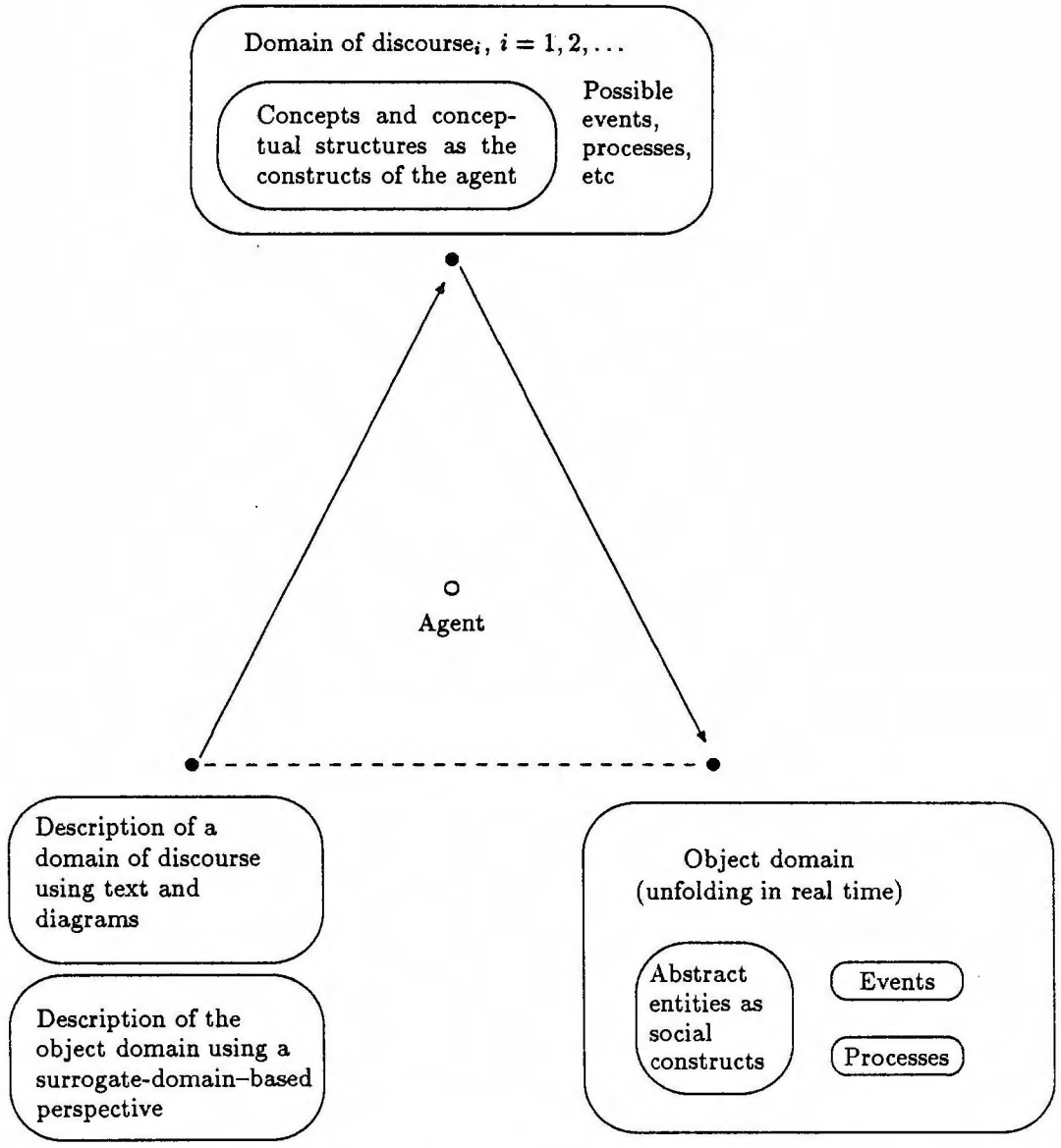


Figure 10: Meaning triangle of domains.

10.1 Sortal non-specificity

We consider the meaning triangle as a diagramming technique on par with structure charts or dataflow diagrams. In talking about an entity X being "on" a node or vertex of the meaning triangle we are using a spatial metaphor. If X is on vertex A then it cannot be on vertex B. This forces us to "sort" X, determine its ontological status, label it, and place it (by diagramming) on one and only one of the vertices.

During the acquisition interaction, one often encounters sortal vagueness and non-specificity. People insist that X is both a lexical object and a non-lexical object. This category mistake can be handled by using the meaning triangle to construct homomorphic images, linked by coreference links and located on different vertices. As an example, consider the well-known safety slogan:

> Accident is just a word until it happens.

This mixes an event (referent type) with the word "accident" (a word type) with people not having a proper conceptual grasp of the notion of an accident (concept type). We may diagram "accident", [ACCIDENT], and the event-of-an-accident or the public-concept-of-an-accident on the word, concept, and referent nodes respectively.

10.2 Distinguishing concepts and words

The knowledge acquisition predicament (Regoczei and Hirst, 1988) is this: The analyst is after knowledge, but all he gets are words. Generating concepts under the input stimulus of words is a laborious process that is often not executed. Words are just accepted and used as empty tokens, with no thought to the concepts "behind" the words.

Many existing expert systems contain knowledge only at the verbal level. MYCIN (Shortliffe, 1976), for example, deduces character strings from other character strings. Much of the knowledge that is written in logical notation has no concepts behind it.

Separating concepts and words enables one to implement systems with a two-level architecture: verbal and conceptual. Concepts are implementable on a computer as abstract data types.

10.3 Is there a referent?

As the informant is describing his mental models, the most powerful question that the analyst can ask is this: "Is there a referent?" If the informant doesn't understand the question, the analyst can use the meaning triangle to explicate the distinction between the verbal, the conceptual, and the referent. If there is no referent in the object domain—either as concrete, physical objects, or abstract, conceptual entities—the informant is describing his own, idiosyncratic mental models. His description should not be rejected. Part of his expertise may derive from such knowledge.

But it should be clearly labeled as such, and perhaps contrasted with more "official" versions (see Section 8).

11 DRAWING DISTINCTIONS

At times, experts seem to be operating with seemingly inconsistent and contradictory knowledge. The contradictions can often be resolved by drawing appropriate distinctions, readjusting concept cluster attachments, and creating new entities through the construction of new concepts.

11.1 Sortally composite entities

Eddington, in the introduction to his 1927 Gifford lectures (Eddington, 1928), adopted the startling strategy of talking about his "two tables":

> I have settled down to the task of writing these lectures and have drawn up my chairs to my two tables. Two tables! Yes; there are duplicates of every object about me—two tables, two chairs, two pens. (p. xi)

One table is the familiar, commonplace object of naive physics that everyone is, or should be familiar with: it has dimension, colour, substance, and some degree of permanence. The other table is the table of "sophisticated" physics, consisting mostly of empty space between nuclei of atoms and electrons moving about with great speed. Yet, despite this strange construction, this scientific table was able to support Eddington's elbows and his writing paper. The puzzling feature is that there is only one referent object, but two conceptual worlds. Is Eddington talking about one "real" table, or two "real" tables?

To resolve the inconsistency, we have to conclude that in talking about "two tables", Eddington is not talking about the table pictured on the referent node. Rather, he is talking about composite entities consisting of a referent physical object and a concept cluster attached to it. This composite entity we could describe as an ordered pair:

(physical referent object, concept cluster)

Attaching concepts to words and referents creates composite—sortally composite—entities. In many cases the use of knowledge, the use of language, and the manipulation of objects in the physical world involve composite entities.

A common misunderstanding is this: The expert seems to be talking about the referent object, but in fact he is talking about *several, different* sortally composite entities, with the same first component.

11.2 Attaching concept clusters

As an example of knowledge acquisition for an expert system for identifying chemical spills and their sources (Buchanan et al, 1983: 134), the following dialogue fragment is reported:

Knowledge Engineer: Suppose you were told that a spill had been detected in White Oak Creek one mile before it enters White Oak Lake. What would you do to contain the spill?

According to the report, the obliging expert proceeds to consider and describe how to find the source and determine what the spilled material is. But let's imagine a more interesting turn in the conversation, one that a skeptical, more battle-scarred expert might take:

Expert: Well, wait a minute now. Before I "contain the spill" I have to verify that the report is accurate. Just because somebody claims that there's a spill, that's not going to make me jump to conclusions. How does he know? Do I believe him? He detected some foreign substance. Fine. Then that's what he should report.

KE: But suppose it really is a spill. He sees it pouring out of a drum or a tank.

E: I would ask him to turn off the tap. To find out what the substance is, he should check the label on the drum to see what it contains. It might be lemonade.

KE: That doesn't take much expertise.

E: Right. The trick is to figure out what people mean when they say things. People attach all sorts of strange concepts to what they see. As Whorf pointed out some time ago, empty drums can be quite dangerous.

The skeptical expert has a point. The trick is to associate with the empirical observation a concept cluster centered on the concept [SPILL]. The expert is drawing a distinction between a spill (a physical referent, "out there") and the verbal report of a spill. He asks how the person making the report attached concepts to what he experienced or observed (right leg of the meaning trapezium). He is also reluctant to immediately attach the general concept [SPILL] to a verbal report containing the word "spill" without receiving further information (left leg of the meaning trapezium). This point can be made even more strongly by recalling recent events in the Persian Gulf involving the *USS Stark* and the *USS Vincennes*, where in both cases the labels of "friend" or "foe" were wrongly attached to external referents.

11.3 Creating new entities by constructing concepts

Hybrid entities are composed of public concepts, text, and hardware. We can create new hybrid entities by constructing concepts, turning them into social constructs, and projecting them onto the physical part of the referent object domain. In the five-chairs example, a group of chairs was

created by attaching the concept [GROUP-OF-OBJECTS] to the chairs already in the room. We grouped the chairs by attaching a concept. It is this concept that gave added structure to the contents of the room. We did not create chairs as physical objects.

Creating corporations, academic disciplines, nations, states, and countries as hybrid entities is done by constructing the appropriate concepts and projecting them onto the object domain. Creating sortally composite activities by concept cluster attachment was already discussed in Section 11.1.

12 CONCLUSION

Informant-analyst–based, natural language–mediated knowledge acquisition interaction is facilitated by the use of diagramming techniques such as the meaning triangle that are acceptable to both informant and analyst. These techniques help the largely human-based acquisition process.

It is reasonable to think that future developments of computer-based techniques would be founded on a better understanding of people-based techniques. Thus the use of the meaning triangle described in this paper is not only to help build better knowledge-based systems now, but also to chart a migration path for computer-based knowledge acquisition. Focusing on concepts, rather than propositions, facts, or rules enables meaning triangle techniques to be integrated with other techniques using personal constructs, the repertory grid, and protocol analysis (Boose, 1984; Gaines, 1986; Shaw and Gaines, 1986).

13 EPILOGUE

After reading this paper, people may still ask (trying to do knowledge acquisition of sorts): "This is great, and it reads well, but what *is* a concept?" We may demonstrate the techniques outlined in this paper by trying to handle this question.

First, the word "concept" may be looked up in the *Oxford English Dictionary*, because that is how one finds out about words.

Second, the concept of concept, the reader either grasps or does not grasp. Concepts have to be created by the agent; they cannot simply be handed over by another.

Third, the concept as referent is tricky. The authors of this paper testify that they have a fairly clear notion of the concept of concept. In fact, they have quite a few different conceptions of the concepts of concept. These concepts are on the referent node relative to the reader. As for a public, abstract concept of concept, it is one of the most basic, underlying concepts of Western thought.

This is how a reflective practitioner would handle the question. It is this approach, demonstrated in this epilogue, that helps the reflective practitioner avoid the paralysis generated by sterile disputes unconstrained by the operational necessity of creating working software.

ACKNOWLEDGEMENTS

We are grateful to John Burbidge, Roy Hagman, and Ed Plantinga for discussions, and to an anonymous referee for his or her suggestions. This work was supported by a grant to the first author from the Trent University Committee on Research, and to the second author from the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

Boose, John H (1984). "Personal construct theory and the transfer of human expertise." *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, Austin, August 1984, 27–33.

Buchanan, Bruce G; Barstow, David; Bechtal, Robert; Bennett, James; Clancey, William; Kulikowksi, Casimir; Mitchell, Tom; and Waterman, Donald A (1983). "Constructing an expert system." In: Hayes-Roth, Frederick; Waterman, Donald A; and Lenat, Douglas B (editors). *Building expert systems* (Teknowledge series in knowledge engineering). Reading, MA: Addison-Wesley. 127–167.

Eco, Umberto (1976). *A theory of semiotics.* Bloomington: Indiana University Press.

Eddington, Arthur S (1928). *The nature of the physical world.* Cambridge University Press.

Gaines, Brian R (1986). "An overview of knowledge acquisition and transfer." *Proceedings, Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November 1986.

Martin, Graham Dunstan (1975). *Language, truth, and poetry: Notes towards a philosophy of literature.* Edinburgh University Press.

Ogden, C K and Richards, I A (1923). *The meaning of meaning.* NY: Harcourt, Brace, and World. 8th edition, 1946.

Regoczei, Stephen and Hirst, Graeme (1988). "Knowledge acquisition as knowledge explication by conceptual analysis." In: Sowa, John F; Foo, Norman; and Rao, Anand (editors). *Conceptual graphs for knowledge systems, Volume I: Knowledge representation*, Reading, MA: Addison-Wesley, 1988 [to appear].

Regoczei, Stephen and Hirst, Graeme (forthcoming). "Knowledge acquisition and requirements specifications: Uses of the meaning triangle." In preparation.

Regoczei, Stephen and Plantinga, Edwin Peter Owen (1987). "Creating the domain of discourse: Ontology and inventory." *International journal of man–machine studies*, **27**(3), September 1987, 235–250.

Schon, Donald A (1983). *The reflective practitioner: How professionals think in action.* NY: Basic.

Shaw, Mildred L G and Gaines, Brian R (1986). "Techniques for knowledge acquisition and transfer." *Proceedings, Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November 1986.

Shortliffe, Edward Hance (1976). *Computer-based medical consultations: MYCIN* (Elsevier computer science library; Artificial intelligence series 2). New York: Elsevier.

Sowa, John F (1984). *Conceptual structures: Information processing in mind and machine.* Reading, MA: Addison-Wesley.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

WIDENING THE KNOWLEDGE ACQUISITION BOTTLENECK

Celine Rouveirol

Laboratoire de Recherche en Informatique,
U.A. 410 du CNRS
Bât 490, Université Paris-Sud
91405 Orsay Cedex, France
email : celine@lri.lri.fr

Michel Manago

INTELLISOFT

28, Av Georges Clémenceau
91400 Orsay, France

Abstract : This paper presents the KATE™(Knowledge Acquisition Tools for Experts) system. The system applies knowledge intensive induction techniques which have been proven effective in real world domains to Knowledge Acquisition for Knowledge Based Systems. KATE is able to use background knowledge about the domain during the learning phase. Given examples of the expert's diagnosis and background knowledge about the application domain, the system automatically generates decision tree as well as expert rules for all the classes of examples at once.

I. INTRODUCTION

Formalizing the knowledge needed to solve a real world problem is far from being a trivial task. As noted in the first AAAI workshop on Knowledge Acquisition, KA is not 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.Thus, a human expert who might possess a ver efficient mental model of his application domain is not always capable of building a computable model that achieves the desired purpose. Moreover, the expert may not be the only source of information : world observations and formal knowledge about the domain may be available. All these provide information which is potientially incorrect,

incomplete or even contradictory. By widening the knowledge acquisition bottleneck, we mean providing sofware tools that take into account different sources of information to build a workable model. In this respect techniques in machine learning that have been studied for over a decade are of particular value when inserted into the knowledge acquisition process.

The work presented in this paper addresses the issue of using machine learning techniques to smoothe and improve the knowledge acquisition step. We first examin the differences between the classical manual methods of knowledge elicitation and the automatic machine learning alternative illustrated by our knowledge intensive learning system KATE™ (Knowledge Acquisition Tool for Experts). We then describe some technical aspects of the system and relate them with the current state of the art in Machine Learning.

In this paper, we will not describe how the system assists the user in interactively building a questionnaire, nor how it learns knowledge from the training examples used against noise and how it uses statistical pruning techniques (see [Manago & Kodratoff, 1987]).

II. KNOWLEDGE ELICITATION VERSUS KNOWLEDGE RECOMPILATION

A major drawback of knowledge elicitation is that it relies on the expert's ability to explain or even trace his own reasoning and behavior. However, human experts are trained to solve practical problems and not necesseraly to explain how they solve them. Even in the ideal case where a well defined theory of the domain is available, the human expert does not always react or reason according to this model, often prefering very specific heuristics to more general procedures. These heuristics constitute the expert's original skill and improve his performance. Complex skill is acquired through theory but also experience and how to model it is still one of the biggest problem of KA. If no nicely tailored theory is available (i.e. in most of the real world applications), experts are then bound to reason using this set of heuristics which differ sensibly from an expert to another... Moreover, one may argue about the adequacy of the production

rule formalism to model the expert's explanations when trying to build a model for the first time.

Considering these strong constraints, it seems obvious that the expert in his first attempt to build a model gives incomplete and / or contradictory rules and then spends a lot of time debugging it. Assuming he suceeds, the model, although correct, is only representative of the single expert who has built the knowledge base; two different experts almost never agree on the rules that should be in the knowledge base. This is the source of problems, in particular those connected with expert system's maintenance.

Machine Learning (ML) techniques, and especially the so-called "inductive" or "learning from examples" techniques offer an alternative to the traditionnal method. Given a set of examples of expertise, the inductive learning system automatically computes rules that model this behavior. The immediate consequence from a knowledge acquisition point of view is that instead of providing rules, which are already synthetic and high level knowledge, an expert only has to provide examples, which are a simpler and more accessible information. For instance, instead of asking a medical doctor to formulate rules to diagnose specific diseases, we ask him to provide examples of the diseases (i.e. a list of symptoms together with the associated diagnosis) and let the inductive learning system generate the rules.

Viewing the examples as unstructured low level information and rules as general concepts extracted from the examples, inductive learning can be seen as *knowledge recompilation* as opposed to knowledge transfer. Inductive learning tools help the expert organize knowledge and formalize rules. He is asked to provide information that he is used to handle and that is therefore easier for him to apprehend. In this sense, Machine Learning techniques allow the expert to concentrate on the most "human-oriented" steps of KA : the expert still plays a critical role since he must provide the vocabulary needed to describe the events (the language of description), the general knowledge about the domain and he must validate (sometimes provide) the set of training examples. After the automatic learning session, the set of rules provided by the system is then tested against the expert's advice, who can refine them at will.

Several scientific experiments have shown that better results can be obtained through the use of inductive tools than through knowledge elicitation. Empirical comparisons were made between expert systems that were built through expert's interview and expert systems built using inductive learning tools. The results demonstrated quite clearly that expert systems are generated faster using Machine Learning, and that the final systems provides often more accurate results than the ones generated through knowledge elicitation approach (see for instance [Michalski & Chilauski, 1980] {Quinlan, 1986]).

III. A DIFFERENT METHODOLOGY FOR KNOWLEDGE ACQUISITION

Before detailing the KA methodologogy inferred by the use of machine learning techniques, let us first elicitate the different kind of information present in any KA process.There is first the vocabulary of the application. This is a very crucial choice that influence all the KA process, as the choice of terms to descibe the examples clearly introduce a bias on what the system is able to represent and then to learn [Utgoff, 1986]. Inductive learning systems use as well domain dependent information than can be split in two : the background knowledge and the example base. Building a parallel between the learning system and the brain, these are the memory of the system. The background knowledge is the long term memory (or semantic memory [Sowa, 1984]), containing common sense information, storing regularities of the domain, such as prototypical descriptions of objects and observed relationships between these objects The example base is the short term memory (or episodic memory [Sowa, 1984]), storing the translations in the representation language of observed specific cases of the domain. The result of the KA process is a set of rules synthetising the knowledge contained in both background knowledge and example base according to the goal affected to the system to be build, such as classification or planing. One of the main features of KATE is its ability to use background knowledge (also called half order theory [Fu & Buchanan, 1985]) and not only rely on statistical regularities in the examples. This makes KATE a *knowledge intensive* inductive learning system.

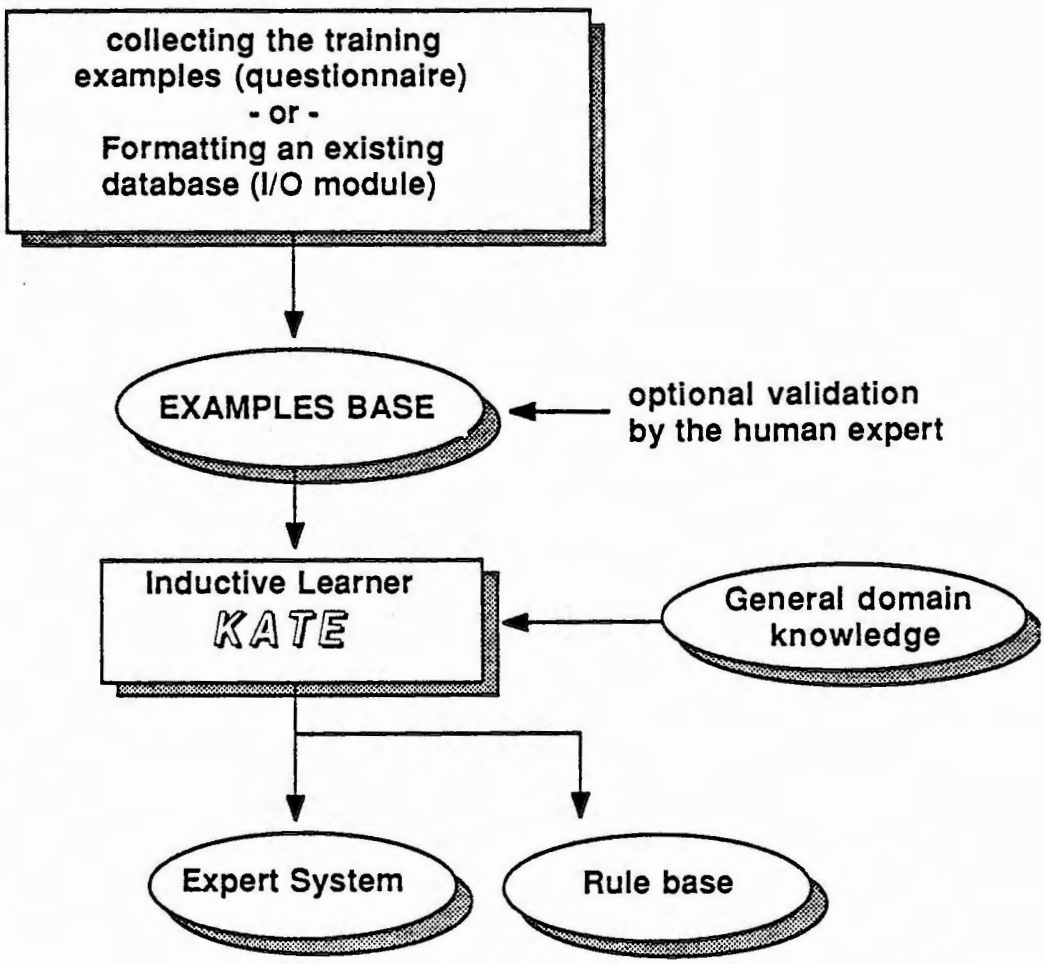


fig1. Methodology of Knowledge Acquisition using Machine Learning

As pointed out before, inductive learning technology can drastically help building an expert system. However, it is not selfstanding as far as KA is concerned. KATE does not learn initial knowledge which has to be provided by the expert, as in most of the inductive learning approaches. There are nevertheless some inductive learning systems that refine their language of representation by introducing new intermediary terms in the vocabulary (shifting biases [Utgoff, 1986], symbolizing taxonomy points [Fu & Buchanan, 1985], creating new frames [Manago, 1986]). Choosing the representation language and the vocabulary of the application are indeed inter-twinned

problems. The languages for input (examples), background knowledge, output (set of rules) are not necesseraly the same. For example, the input language may not accept variables, as the output language does. The choice of logic to be used is highly application dependant and relies on practical considerations such as the need to represent high level concept involving several objects for the background knowledge, the need to internally use quantified variables and so on... Using a feature-value pair language, the system must be given valid attributes names and appropriate ranges for the attribute values. Using typed first order logic, the system needs prototypes, attributes and maybe allowable values for the attributes. In the hypothesis of using first order logic, the choice of what should be a constant or a predicate is as well far from being trivial (among all the pieces of low level information, predicates are sort of priviledged information). Moreover, it must be kept in mind that a trade-off between generality and efficiency of the representation has to be taken into account. Inductive learning from examples tends to be an explosive process unless severely constraining the space of possible inductions from a set of examples, and one solution to decrease the complexity of learning is reducing the language to a carefully chosen subset of first order logic if not to propositional calculus.

There is no universal method to obtain the initial vocabulary, or set of descriptors. Identifying low-level concepts may be from fairly easy (as it is for an application with strong domain theory) to extremely difficult (for example in air traffic control, it is difficult for the expert to express, let it be in natural language, the relevant knowledge contained in a radar picture). The descriptors (or low level concepts) can be gathered by searching the literature, by interviewing an expert and translating natural language into the proper formalism (see for example [Regoczei & Plantinga, 1986]), or by using an interactive tool. On the one hand, the choice of descriptors depends on the underlying logic used by the system : for example, with propositional calculus, the three descriptors BLUE, SQUARE and BLUE-SQUARE are needed to describe the concept of a blue square, instead of two predicates BLUE(x) and SQUARE(x) in first order logic. On the other hand, it depends, within the same domain, of the accuracy needed to correctly describe the examples and the valid primitives on those examples. For example, in the world of blocks, if the application is to move blocks around (as in SHRDLU [Winograd, 1972]), CUBE is a valid primitive. But if the system is to perform analysis of visual scenes in that same world of blocks, the concept of CUBE

will be described in term of concave / convex, interior / boundary lines ([Waltz, 1975]) and CUBE will therefore be a high level concept .

We address with more details the representation issues we met, but let us say that we chose an *frame based representation* for the background knowledge, allowing to efficiently store the static information about observed regularities of the domain as well as axioms use to rewrite or to complete examples description if needed and *first order logics* for the representation of the examples and of the generated rules.

Given this basis, one can easily start building on-line questionnaires to collect the examples and to gather information for the background knowledge. This domain specific knowledge takes the form of :

- *prototypical objects* (elephant is a specy, elephants are mammalians and vegetarians, one part of the elephant is the trump...)
- *default properties* (elephants are ususally gray)
- *axioms* which allow to deduce low level concepts from other concepts, how to deduce the age of an elephant according to its appearance).
- *constraints* on values of properties (elephants can only be gray white or pink)

Finally, the training example base must be provided to KATE. When there is an existing database (for example, hospital records), the traces of the consultations can be recorded and validated by the expert (as done in [Quinlan, 1986]). We can also ask the expert to provide examples of consultations based on observed cases by filling up questionnaires. Note that questionnaires are only used as media to record actual examples of the expert in action and produce training examples for the learning system. The example base must be representative of all the domain, which means that the example base has to cover the hole domain in order to provide a correct rule base.

IV. THE LEARNING COMPONENT OF KATE

IV.1. HEURISTIC SEARCH

KATE is based on a inductive learning algorithm ID_3 first described in [Quinlan 1983]. Given a set of examples that may belong to a number of different classes, it produces a decision tree that discriminates between the classes in the most efficient way. Note that this induction tree can be translated in the form of classification rules ([Corlett 1983] and [Quinlan 1987]).

Let us first briefly review the heuristic search procedure that is common to all these systems. Typically the system outputs a decision tree which enables classifying an unknown example by asking the fewest number of questions. The system has to find at each step of the tree building procedure which test is maximally discriminant (i.e. that best separates the elements of each class). This operation relies on a heuristic evaluation function, called *information gain* that estimates the amount of information still needed to classify the examples at a given node of a tree (this measure is the entropy measure, [Shannon & Weaver, 1949]). For more details about the different measures used to build the induction tree and their respective influence on the shape of the tree, refer to [Quinlan 1983] and to [Cestnik et al., 1987].

Thus, the system performs a hill-climbing search of the domain [Nilson 1980], using information gain as a heuristic evaluation function. The important point is that this enables us to treat problems of exponential complexity with linear complexity methods. The method is therefore well suited to handle very large problems (i.e. with a large number of examples, attributes and attribute values). Of course, if the heuristic function does not give a good approximation of the shortest path to a solution, we will get results that might not be optimal. However, systems that use information gain have demonstrated that, they achieve good results in practice. These kinds of induction systems have been able to build expert systems in large domains such as chess end-games [Quinlan, 1983], medecine (rules to diagnosis thyroid diseases from a set of 3000 examples described by 29 attributes [Quinlan, 1986], primary tumors, breast cancer, hepatitis, lymphography from a set containing 1000 examples described by 80

attributes [Cestnik et al, 1987]), oil prospection, etc... and led to numerous industrial applications.

IV.2. REPRESENTATION ISSUES

The main advantage of KATE over ID_3-like systems lies in the knowledge representation it uses. ID_3-like systems use feature-value vectors (propositional calculus) while KATE handles an object oriented representation (a form of first order logic [Nilsson 1980]). Let us consider examples that demonstrate limits of the attribute value representation formalism. One of them is the following : it is absurd to consider the color of a spot when there is no spot. Using the feature-value representation, we must have the feature [exist-spot] and another [color-of-spot]. When [exist-spot = no] the attribute [color-of-spot] is not applicable. As object oriented representation is a structured one, this problem is automatically solved : one must access the frame "spot" before accessing the slot "color", which means that color does not exist independently from the object it describes. Another problem with feature-value representation is that we can be interested in the color of spots in general without adressing a particular spot . This addresses the necessity of quantification over object's property. We could reasonably wish as well to code taxonomies (for example "white and yellow are light colors") or other relationships between objects (such as "part-of" relationship).Another limitation of ID_3-like system is the feature independance assumption which is not realistic in pratical situations. This limitation has to be overriden using different methods to build the classification.

KATE uses a representation language combining both object oriented and first order logic layers.

To represent the examples and the rules, KATE uses an efficient object-like representation. It has two advantages over pure object oriented representation : it uses quantified variables and logical connectives : conjunction, negation and internal disjunction. Universal quantifiers are provided by rewriting statements such as "all the elephants are gray" as "∃x [x: <ISA ELEPHANT> <COLOR GRAY>] & ($NOT ∃y [y: <ISA ELEPHANT> ($NOT <COLOR GRAY>)])" or in other words, "there exist gray elephants and no elephant which is not gray"). The format of the examples is

similar to the format of the generalized examples (i.e. rules). The examples are thus logical patterns that bear relationship with one another.

The frame layer is used for representing background knowledge, as all this part of memory is organised according to type or class belonging. The object layer, as a structured representation language, allows to efficiently represent taxonomies or any kind of relationship over objects, attributes and even attributes values. Thus, the system is able to wonder about the information gain of tests such as "Does the object x has a light color?". This is achieved by first wondering about the existence of the object x, and then about its attributes (we will never compute information gain of "Does the object x has the color yellow" before computing the information gain of "Does the object x has a light color" higher up in the tree). This design choice was made to preserve the model-driven, general to specific search properties of the learning system. The object oriented language we use, GOL, was concieved to be as customizable as possible. The user can easily redesign most of it (for example, the name of the facets, the way inheritance and generalization are performed, etc...) according to his convenience. We present here the options which are available by default in the language.

Each generic object (frames [Minsky, 1975]) have a set of slots, with facets and values for the facets. Each facet and facet value correspond to one particular aspect of the property : $value for the actual value of the slot, $if-needed for a procedural call to compute the value of the slot when it is missing and the system attempts to read it, $default for the default value of the slot, $range for constraining the possible values for the slot. Inheritance is performed by looking first at the $value facet, then the $if-needed facet, then the $default and if all fails then go on to the next ancestor frame for inheritance and iterate. Paths are provided to the inheritance function, specifying which links are to be followed to reach the ancestors of an object. Unlike languages which uses recursive inheritance (look for the property in the object, if it is missing, compute the superclass, go there and look for the property again, and iterate), KATE compute all the superclasses relatively to the original one. A path (i.e. a list of links the function has to follow when going though the lattice over a node) is given to the inheritance function.

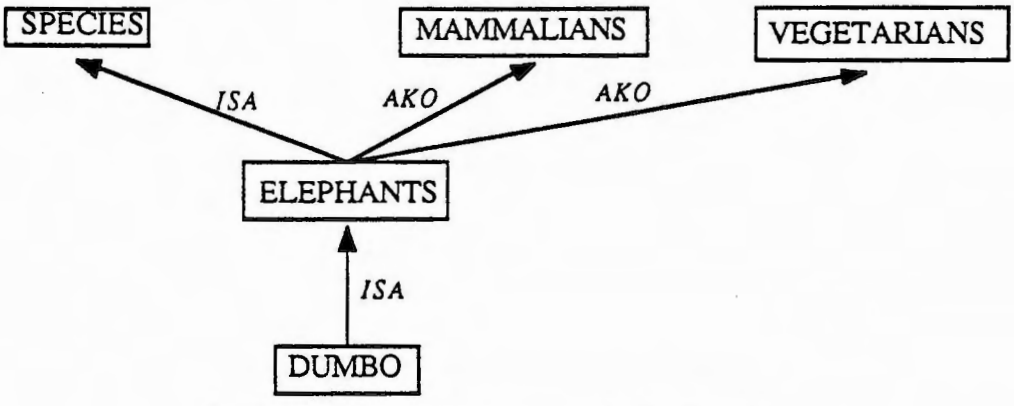


fig. 1 A simple taxonomy represented using frames

In the figure 1, if we specify the inheritance path (ISA ($Follow-several AKO)), DUMBO will inherit from the classes ELEPHANTS, MAMMALIANS and VEGETARIANS and not from the meta-class SPECIES (this would have been the case if we had computed inheritance recursively). The language is also able to manage multiple inheritance and therefore to cope with conflicts when they occur by specifying search methods (i.e. depth-first search, breadth-first search or any user defined search method) in the lattice of relations linking the objects. The first ancestor encountered will have higher priority for inheritance.

Taking into account further improvements of the learning module (see [Manago & Blythe 1988]), this language is also especially adapted to learning and provides primitives to manage generalization, which is indeed the type of learning we are interested in. Generalization as well as inheritance are seen as going through the lattice of ancestors of the objects. Thus, a path and a search method are also specified to the generalization function. For instance, let us assume the inheritance path is (ISA ($Follow-several AKO)) (i.e. follow one ISA link (set membership / class-instance / meta-class) and / or any number of AKO links (set inclusion / superclass) and the generalization path as ($Follow-several ($Follow-either PART-OF AKO)). The important things this example demonstrates are that the system is able :

. to generalize over links different from the usual AKO link (in this examples over part of link : a yellow leaf, a yellow fruit will generalize to part of the plant is yellow)

. to generalize and inherit following different links, giving different semantic flavor to the two of them (it is not because a house is white that all its subparts are white... For instance the roof may be red).

IV.3. TECHNICAL ISSUES OF THE LEARNING SYSTEM

There are several problems connected with designing an ID_3-like system using first order logic (object representation). First, it must use a more elaborate pattern matcher for the counting procedures to evaluate information gain. It must also be able to carry the bindings of the variables through the decision tree and take into account the fact that several objects may match a given test in a specific example. For instance, if we are classifying trains and ask "Is there a red car" and there are several red car, the system must not build any assumption about which car matched. Hence,when asking "is the red car big?", the system should not fail if the first car that matched the pattern is small and there is another red car that is big. In fact, partly because of this possibility to match patterns in several ways, the final structure produced ends up being a lattice and not a decision tree.

The system must also know which tests are relevant. Using feature-value vectors it was quite easy to know what attributes should be considered for a test, as all the relevant information is present in the example description. However, using our object oriented language, the structure of the example can be different from one example to another (for example, properties or subparts may be misssing in some examples' descriptions). Then, how can the system know that asking for a special attribute is legitimate ?

We are currently providing several learning methods to choose the best tests when building the decision tree. The first one, which sticks to the method used in ID_3, consists in dynamically building a lookup table of all the possible tests that can be performed and compute information gain on these. The second one consists in building a "star" (see [Michalski, 1983]) around randomly selected examples and use each complexes to build the tests in the decision lattice. We are also experimenting with Structural Matching [Kodratoff & Ganascia, 1986] to generalize the examples class by class, put them in Structural Matching and use the difference set to build the tests. The

different methods seem to give very different results and the user can select the method that gives the best result for his application (after empirical experiments).

The system has been designed to use, at will, a number of preference criteria for building the tree, including information gain and a chi-squared test of the reliability of an attribute. The user is also able to input his own preference criteria if he wants it. The tree can be terminated at any point if its overall reliability, based on a chi-squared test, falls below a certain value called *min-branch-safety* [Corlett, 1983]. Thus the conclusions of a tree can be "fuzzy" and terminate before an exact classification has been reached.

It has been argued that logical rules are easier to understand than decision trees [O'Rorke, 1982]). By taking the decision lattice that it generates, KATE separates it into its component rules. These are conjunctions of tests along with their conclusions. Each rule is then simplified as follows [Corlett, 1983]: each test of the conjunction is experimentally dropped. If the new conclusion based on the examples that satisfy the new rule is weaker than the original one, the test is put back in. Otherwise, it is dropped permanently. Thus all tests in a rule that were originally in the tree but not directly associated with that rule's conclusion are dropped.

These rules can be given a "flat" representation, understanding that all the descriptors of their Left-Hand Side must simultaneously take the value TRUE to allow the recognition of a high level concept. On the other hand, one might consider that some descriptors are more important or better defined or less noisy, or more efficient than others. One can then take advantage of this knowledge by checking the better ones first (meta-level knowledge). This choice clearly influence the recognition rules.

V. CONCLUSION

We have presented in this paper the system KATE which uses knowledge intensive inductive learning techniques to aid the knowledge acquisition process. It integrates several learning methodologies such as Quinlan's ID_3, Michalski's INDUCE, Kodratoff's Structural Matching and benefits from a powerful object oriented environment. This tool has been designed to have an open architecture and to use domain knowledge when building classification rules. It is efficient on large databases

when a large number of representative examples of expertise are available (like in ID3, the complexity of the method is linear with respect to the number of examples). Since it uses background knowledge through the object oriented mecanism, it is expected to be well suited for complex real life domains which could not be tackled with existing inductive tools up to now.

KATE can generate running expert systems and is thus a self contained product: knowledge acquisition phase (inducing the decision tree) + inference phase (exploitation of the decision tree). Moreover KATE can also generate production rules or object rules from the decision tree. It can therefore be used as a Knowledge Acquisition front-end for an existing inference engine.

ACKNOWLEDGEMENTS :

KATE is an internationally registered trademark by INTELLISOFT inc. Some of the ideas that were developped in KATE originated from the ESPRIT project INSTIL P1063 [Kodratoff et al, 1986]. None of this work would have ever taken place without the support from the European Economic Community, from the Greco and PRC "Intelligence Artificielle" and from the partners of INSTIL: GEC-research, the Laboratoire de Recherche en Informatique in University of Orsay and Cognitech. We would like to express our deepest gratitude to all that made valuable contributions to this research: Yves Kodratoff, Jim Blythe, Dave Neeham, Clive Smallman, Rob Corlett, Patrick Corsi, Thierry Andro, François-Marie Lessafre, Marco Valtorta, Roberto Desimo and Martin Van Somerhen. We would also like to thank Bruce Buchanan and the Helix group at the Knowledge Systems Laboratory (Stanford University) for welcoming Mr Manago in their lab and for all the precious comments and advices they gave him so that the system could be improved. KATE is implemented in Common-Lisp and runs on the EXPLORER™Lisp workstations and on the Micro-EXPLORER™ from Texas Instruments.

REFERENCES

(Bareiss et al., 1987) Bareiss E.R., Porter B.W., Wier C.C., "PROTOS, an Exemplar-Based Learning Apprentice," *2nd AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems*, Banff (Canada) 1987.

(Cestnik et al, 1987) Cestnik, B. & Konokenko, I. & Bratko, I. ASSISTANT 86: A Knowledge-Elicitation tool for Sophisticated Users, in Bratko, I. and Lavrac, N. (eds), *Progess in Machine Learning*, Sigma Press : Wilmslow.

(Corlett, 1983) Corlett, R.
"Explaining induced decision trees," *Proc. Expert Systems*, pp. 136-142, 1983.

(Feigenbaum, 1981) Feigenbaum, E. A.
"Expert Systems in the 1980s," in Bond (ed), *State of the Art Report on Machine Intelligence*, Maidenhead: Pergamon-Infotech, 1981.

(Fu & Buchanan, 1985) Fu, L. M. & Buchanan B. G.
"Learning Intermediate Knowledge in Constructing a Hierarchical Knowledge Base", *working paper N° KSL 85-4*, Computer Science Deparment, Stanford University, 1985.

(Kodratoff et al, 1986) Kodratoff, Y. & Manago, M. & Blythe, J. S. & Smallman, C. & Andro, T.
"Generalization in a Noisy Environment: the Need to Integrate Symbolic and Numeric Techniques in Learning," *Proc. of the 1st AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems*, Banff (Canada) 1986.

(Kodratoff & Ganascia, 1986) Kodratoff Y., & Ganascia J.G.
"Improving the generalization step in learning", *Machine Learning 2 : An Artificial Intelligence Approach,* Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann, Los Altos 1986

(Kodratoff & Tecuci) Kodratoff, Y. & Tecuci G.
"What is an explanation in DISCIPLE", *Proc. of thr Fourth International Workshop on Machine Learning, Irvine (Cal) 1987*

(Manago, 1986) Manago, M.
"Object-oriented Generalization: A Tool for Improving Knowledge Based Systems," *Proc. of the 1st International Meeting on Advances in Learning*, Les Arcs (France) 1986.

(Manago & Blythe,1988) Manago M. , Blythe J.
"Learning disjunctive concepts", *Proc. of KROML workshop*, to be published in *Lectures notes in Computer Science, Springer Verlag.*

(Manago & Kodratoff, 1987) Manago, M. & Kodratoff, Y.
"Noise and Knowledge Acquisition," *Proc. of the 10th International Joined Conference in Artificial Intelligence (IJCAI 1987)*, Morgan-Kauffman : Los Altos (CA), 1987.

(Michalski & Chilauski, 1980) Michalski, R.S. & Chilauski, R.L.
"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," *Policy Analysis and Information Systems*, Vol 4:2, 1980.

(Michalski, 1983) Michalski, R. S.
"Theory and Methodology of Inductive Learning," in *Machine Learning 1: An Artificial Intelligence Approach*, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann, Los Altos 1983

(Minsky, 1975) Minsky, M.
"A framework for representing knowledge," In The Psychology of Computer Vision, Winston P. H. ed, Mc Graw-Hill, New York 1975.

(Nilsson, 1980) Nilsson, N.J.
Principles of Artificial Intelligence, *Morgan-Kaufmann*, Los Altos (CA), 1984.

(O'Rorke, 1982) O'Rorke, P.
"A Comparative Study of Inductive Learning Systems AQ11P and ID3 Using a Chess End Game Test Problem," *Report ISG 82-2, UIUCDCS-F-82-899*, Dept of Computer Science, University of Illinois, Urbana, 1982

(Quinlan, 1983) Quinlan, J. R.
"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, Los Altos 1983

(Quinlan, 1986) Quinlan, J. R.
"Simplifying decision trees," *Proc. of the 1st AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems*, Banff (Canada) 1986.

(Quinlan, 1987) Quinlan, J. R.
"Generating production rules from decision trees," *Proc. of the 10th International Joined Conference in Artificial Intelligence (IJCAI 1987)*, Morgan-Kauffman : Los Altos (CA), 1987

(Regoczei & Plantinga, 1986) Regoczei, S. & Plantinga, E.O.
"Ontology and Inventory: A Foundation for a Knowledge Acquisition Methodology,"*Proc of the 1st AAAI Workshop on Knowledge Acquisition for Knowledge Based Systems*, Banff (Canada) 1986.

(Shannon & Weaver, 1949) Shannon & Weaver
The Mathematical Theory of Computation, University of Illinois Press, Urbana : Illinois, 1949.

(Sowa, 1984) Sowa J.F.
Conceptual structures, Addison Wesley, 1984

(Utgoff, 1986) Utgoff, P.E.
"Shift of Bias for Inductive Concept Learning," in *Machine Learning 2: An Artificial Intelligence Approach*, Michalski R. S., Carbonell J. G., Mitchell T. M. eds, Morgan Kaufmann, Los Altos 1986.

(Waltz, 1975) Waltz, D. L.
"Understanding Line Drawing of Scenes with Shadows," in *The Psychology of Computer Vision*, Winston P. H. ed, Mc Graw-Hill, New York 1975.

(Winograd, 1972) Winograd, T.
Understanding Natural Language, Academic Press, New York 1972.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

HOUSE CALLS:
Building and Maintaining a Diagnostic Rule-Base

Kalev Ruberg Steven M. Cornick Kay A. James

Advanced Technology Laboratory
Institute for Research in Construction
National Research Council Canada
Ottawa, Canada K1A 0R6

ABSTRACT

A knowledge-acquisition system was designed and built to help an architectural firm automate their diagnosis of building problems. This system was tailored to the firm's existing data-base system and building survey method. Rules are generated from the data using induction learning algorithms ID3 and AQ11, and the orderly development of the rule-base is ensured by a verification procedure. Architectural diagnostics rely on the expertise of an experienced analyst. Building diagnostic processes require spatial, verbal and numerical reasoning. A rigid data structure imposed by the firm excluded crucial levels of semantic information. Induction methods proved useful tools for organizing data, but the expertise was captured by editing the rule base. This project identified the need in the building industry to develop a taxonomy and a representation to support the building diagnostic process.

INTRODUCTION

Induction (machine learning) algorithms are useful tools for organizing large databases into production rules. However, for these rules to produce meaningful inferences the learning techniques must be augmented by less constrained methods, including manual rule authoring procedures. Rule-bases produced using manual editing facilities require verification and maintenance procedures to prevent contamination of the rule base. A system based on these principles was developed for an architectural firm diagnosing faults in buildings.

Numerous knowledge based system prototypes have been developed for the building industry. Most of these systems have been for diagnosing building problems. Typically, these prototypes focus on a particular aspect of the building. The majority of them are rule based systems and follow a conventional pattern of knowledge acquisition; either the experts themselves have built the systems, or knowledge engineers have constructed the rule base using various interview techniques (Davidson, Davidson and Ruberg 1988). These knowledge acquisition techniques limit the size of prototype systems and their extensibility.

A project undertaken by an architectural firm emphasized the problems of developing a large knowledge base for building diagnosis. The ambitious nature of the project, automatic diagnosis of most building problems, served to illustrate key issues. Several experts would be needed to construct such a knowledge base. The knowledge base would continue to grow as

new illnesses and relationships were discovered. A systematic method of recording data and a *building related pathology*; linking data to specific buildings and building illnesses would be required.

To aid in the generation of a working knowledge base, the development of a rule authoring assistant was undertaken. The methodology was based on incremental expansion of the knowledge base, coupled with rule base verification after each step. The system was built to assist in the construction of a knowledge-based system using the architectural firm's diagnostic techniques. Rules were created by an author and were generated from data using machine learning techniques. The system consists of three components, namely an interface and editor module linking an inductive rule generation system and a rule verification system.

DIAGNOSING BUILDING FAULTS

Diagnostics

Diagnostic methods range from deducing faults from causal models to induction of diagnostic heuristics from empirical associations. Causal models are often based on systems models of physical domains and are used as comparative predictors for events in the real world. For example, a device model of a satellite's attitude controller will provide the diagnostician with an accurate representation of its performance based on equations describing its behaviour (Pazzani 1986). By introducing new parameter values, errant behaviours could be replicated by a knowledgeable user. Causal models are applicable in domains where the physical objects, their behaviour and their interrelationships can be explicitly described in mathematical syntax. On the other hand, domains that rely exclusively on heuristics (such as management) would be described only by observed empirical relationships.

Strategies combining both causal models and heuristics are possible. These are illustrated by approaches used in diagnosing electronic components where partial systems models of power supplies are augmented by heuristic reasoning concerning their faults (Kahn 1987).

Acquiring the diagnostic heuristics includes:

- manual transformation of freely given verbal reports (Hoffman 1987) into formal algorithms or production rules,
- structured data completion and elicitation of associations (Boose 1986)
- recording the diagnostic associations in structured databases (Michalski *et al.* (1986).

In free verbal descriptions, formal diagnostic procedures are described as the gathering of data to support an hypothesis (forward chaining). In practice, a likely hypothesis is postulated and the facts to support the hypothesis are sought. Contradictions lead the practitioner to abandon the hypothesis and embrace a new goal (backward chaining) (Welbank 1983).

In order to be able to automate the diagnostic process, it is necessary to :

- have a taxonomy that supports both paradigms; one for processing (forward) and one for representing the knowledge during acquisition (backward),
- have a representation with a resolution capable of deriving the resultant causes and remedies from symptoms
- have a tractable problem space.

Diagnostic procedures have been explicitly defined in the medical field. Medicine provides a framework for developing pathologies based on a mix of causal and empirical associations. Within the medical domain there is agreement on taxonomy.

Other fields where diagnostic procedures are well established include VLSI and circuit board analysis. In most engineering fields, the diagnostic process is constrained by the operational characteristics of the device. It is often possible to rely on causal models for diagnostic information (Tanaka 1986).

Automating the Diagnostic Process

Pioneering work in the expert system field was concentrated in medical diagnostics. Automated diagnostic systems typically specialize--MYCIN (Shortliffe 1976) for blood infections; NEUROLOGIST-1 (Xiang 1984) for neurological disorders. About 60 others have been described (Waterman 1986). Automated diagnosis is also prevalent in the VLSI industry, satellite trouble-shooting and finding faults with industrial machinery. In medical, industrial engineering and electronics fields, the diagnostic process is well constrained when compared with building related problems. Automated diagnostic systems have typically used production rule representations.

Diagnostics in the Construction Field

Diagnosing faults in buildings is a process similar to medical diagnosis. Symptomatic evidence is used to determine a cause and a related cure. However, there are characteristics about buildings and the related design industry that lead to problems when developing diagnostic systems:

- no accepted taxonomy exists. Loose definitions are used to describe the various parts of buildings,
- there is no "generic" building type. Layout, structure, materials and operating environments are difficult to characterize and classify. (The analogy to medicine would be Dr. McCoy's task of diagnosing the ailments of many aliens without benefit of a pathology or taxonomy).
- spatial relationships between systems are inconsistently described and not defined,
- not only are the states of objects and systems critical, but the links and the state of the relationships between systems are critical as well.
- there is no well defined pathology describing building problems
- information about building illnesses has not been gathered and organized
- building science lacks complete systems models on which causal diagnostic procedures are founded.

Symptoms and signs are recorded by the diagnostician. For the purposes of this discussion, symptoms (reported by the user) will be combined with signs (objective physical evidence including measurements) into one category called symptoms. The symptoms are effectively organized and a diagnosis is made. Diagnosis must be linked to the prediction of future events, therefore a prognosis is required. The prognosis for the building may be good or poor. A decision whether or not to intervene is based on the prognosis. A reconstruction may be prescribed if a decision is made to intervene.

A syntax needs to be developed to relate the problem (illness) to the outward manifestations (symptoms) and their causes. A taxonomy is also needed to ensure that symptoms of problems, building related symptoms, structures, materials, and diagnostic results are recorded consistently. Currently building diagnosis is more of an art than a science (Wilson 1984). There are attempts to establish a building pathology for the correct diagnosis of building illnesses (Bayazit 1984).

The Manual Method Building surveys are performed by experts with many years of field experience. Generally these experts rely on inspection and non-destructive testing techniques to establish a diagnosis and a prognosis. The process begins with a walk-through inspection. During this inspection the expert would record his observations, making notes, taking photographs or video tapes, and perhaps perform some non-destructive tests such as measuring moisture content in wood, measuring crack widths or conducting ultrasonic measurements. An important part of this step is the organization of the information collected in a systematic and comprehensive manner. At this point a mental model is built that combines the condition of the component systems, the symptoms, and the related causes. Eventually all the information would be deductively pieced together and the illness identified.

Automated Diagnostics for Buildings

The current method of building diagnostics requires an expert practitioner. Experts are scarce, and requiring them to perform a time consuming analysis of a building is expensive. An automated method was proposed by an architectural firm in order to document the information used by experienced practitioners and reduce the amount of expertise needed to perform a building diagnosis . The walk through method would be retained; however, a rigid syntax would be imposed on the surveyor. Alphanumeric codes would describe the building systems, construction materials, and observed symptoms and link these with a cause and diagnosis. Quantity of, and percentage of affected materials would also be recorded to automatically provide cost estimates for remedial actions.

Back at the office, the expert would look at the records in the database, complete each record (symptom, cause, diagnosis) and let the induction system look for patterns in the data. The expert would then check the rules, add new attributes if he thought a meta-level was required and verify the manually modified rules with the rule checker.

Rules developed from the database would be posted to the general rule base. When the next database describing a building was given to the expert for analysis, he would allow a knowledge based system (using the generated rule base from previous diagnoses) to infer as many conclusions as is possible with the existing rule base, and then proceed to look for patterns and generate rules from the unused data as in the previous iteration.

Rules in the system are written in a specific syntax unique to the knowledge acquisition and verification module. The syntax is translated to conform to one of three knowledge-based systems shells used to analyze incoming data. The automated process is shown in Figure 1.

Knowledge Representation The attempt by the architectural firm to automate the diagnostic process used a rigid method of describing the building problem. Information at the site was captured by a novice architect using cards as a guide to the required syntax, and recording the findings on Dictaphone tapes. The tape was transcribed at the office into alphanumeric codes representing the object and symptom descriptions. The site description followed an orderly walk around and through the building. As the surveyor gained more experience, the use of codes in place of actual words describing the scene became more common. This description of the building separates the information that will end up in the database from a free form "natural language" description found in the "memo" part of the description. An example description of a concrete building is presented in Figure 2. At this point, the expert must include a cause (and a solution) for each record describing an individual problem. The "memo" field describing the conditions seen at the site was most often used during diagnosis of building problems and completion of the cause and remedy codes in the database.

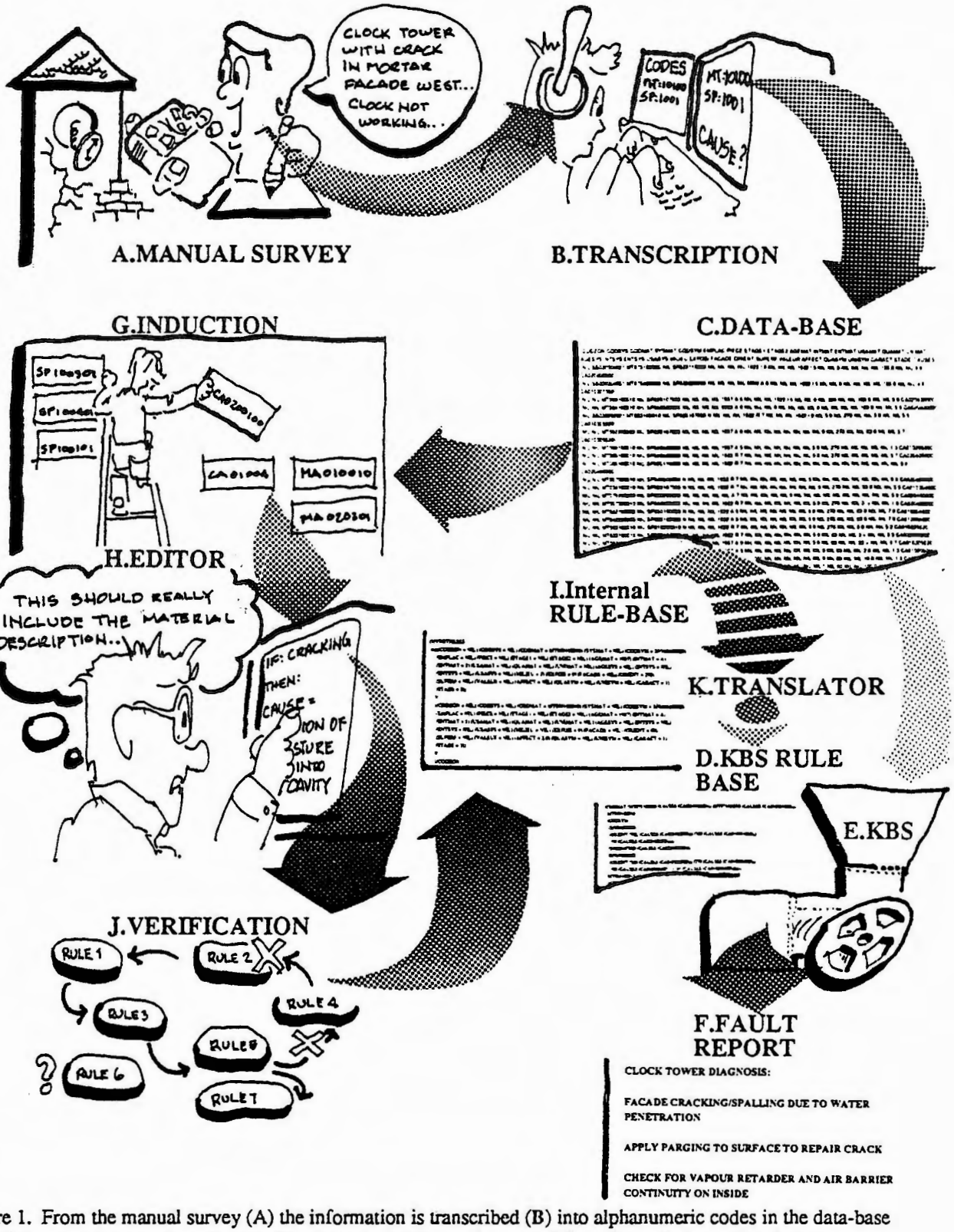


Figure 1. From the manual survey (A) the information is transcribed (B) into alphanumeric codes in the data-base (C). If there are rules in the rule-base (D), the knowledge-based system (E) analyzes the incoming data and produces a fault report (F). The data are also run through the induction learning module (G), rules are output and manually edited (H), posted to the rule base (I) and submitted to the verifier (J). Once through this process, the rule-base is translated (K) to the appropriate KBS syntax.

Zone: *third floor, facade, south-west orientation.*
Memo: *facade has a crack running from the third floor to the second floor*
System code: SS0302000 (looked up in field)
Memo: *stucco on brick facade*
stucco is cracked and spalling
about 80 years old
...

This facade problem would be transcribed as:

Zone: third floor, facade, south-west orientation.
Memo: facade has a crack running from the third floor to the second floor
Systeme code: SS0302000
Memo: stucco on brick facade
Age: 1910
Maintenance:9 (high)
Use: 0 (not operable)
Placement: Part of wall
Exposure: Normal
Material code: MT970200 ...

Figure 2. Format of information developed at the survey site and its transcription at the office. During the second stage, a CAUSE and ILLNESS are associated with the record.

The data structure was designed to describe problems in a building according to general location (zone by floor), material involved (e.g. brick or concrete using a material specification code that is well accepted by the building design community), system in a building (e.g. wall system, mechanical system), the symptom exhibited, and a prognosis for each problem. Each record contained a complete problem description.

The system and symptom descriptions were conceived by the architectural firm. Both the syntax and semantics of these descriptors were first attempts at defining a building taxonomy and pathology. These data would be further transformed to fit the rigid structure of the flat database. At this point it is difficult to read the descriptive evidence presented in each record as all the fields are recoded as alphanumeric codes. An example from a file is shown in Figure 3. (By the end of this project, one diagnostician in the firm was quite capable at reading these alphanumeric codes and describing the contents of each record.)

Knowledge Acquisition The task of building such a "general practitioner" cannot be handled by informal knowledge acquisition methods. An estimate of the number of medium-grained rules in the knowledge base for a competent building ailment diagnostic system (for northern latitudes of North America) exceeded 100,000. This estimate is based on a diagnostic system built for diagnosing window problems. (It contains about 200 rules for diagnosing one of the five ailments in the component. A complete diagnostician for the component might contain 1,000 rules and there are about 50 discrete large-scale components with two systems description levels in most buildings.)

Welbank (Welbank 1983) describes three phases typically followed in knowledge acquisition for knowledge-based system development. The first phase comprises knowledge representation or basic data structure; the second, production of a working system (by the elicitation of knowledge, verification of rule semantics and syntax); and the third, validation or testing and debugging to achieve a level of confidence in the system.

(CODZON CODSYS CODMAT SYSMAT CODSYM EMPLAC PIECE ETAGE1 ETAGE2 AGEMAT INTMAT ENTMAT USAMAT QUAMAT UNIMAT AGESYS INTSYS ENTSYS USASYS MILIEU EXPOSI FACADE ORIENT SUPERF VALEUR AFFECT QUASYM UNISYM CARACT STADE CAUSES)
(NIL SS030304001 MT075100000 NIL SP020110000 NIL NIL NIL NIL 1922 I 9 NIL NIL NIL 1922 I 5 NIL NIL 9 NIL NIL NIL NIL 100.0 NIL NIL 9 8 CA020400000)
(NIL SS030304001 MT075400000 NIL SP030200000 NIL NIL NIL NIL 9999 A 9 NIL NIL NIL 1922 I 5 NIL NIL 9 NIL NIL NIL NIL 100.0 NIL NIL 9 9 CA010301060)
(NIL NIL MT099100010 NIL SP020107000 NIL NIL NIL NIL 1967 A 5 NIL NIL NIL 1922 I 5 NIL NIL 9 NIL 360 NIL NIL 100.0 NIL NIL 5 9 CA020400000)
(NIL NIL MT084100010 NIL SP040600000 NIL NIL NIL NIL 9999 A 9 NIL NIL NIL 0 NIL NIL NIL NIL 9 NIL NIL NIL NIL 100.0 NIL NIL 5 5 CA020400000)
(NIL SS030202001 MT033100010 NIL SP020107000 N NIL NIL NIL 1922 R 7 NIL NIL NIL 1922 I 9 NIL 5 9 NIL 270 NIL NIL 5.0 NIL NIL 5 5 CA010305020)
(NIL NIL MT092200000 NIL SP020107000 NIL NIL NIL NIL 1967 A 9 NIL NIL NIL NIL NIL NIL NIL NIL 9 NIL 270 NIL NIL 80.0 NIL NIL 5 7 CA010305020)
(NIL NIL MT099100010 NIL SP040600000 NIL NIL NIL NIL 1967 A 5 NIL NIL NIL NIL NIL NIL NIL 5 9 NIL 270 NIL NIL NIL NIL NIL 1 3 CA010206000)
(NIL NIL MT085100010 NIL SP040600000 NIL NIL NIL NIL 1922 R 7 NIL NIL NIL NIL NIL NIL NIL 5 9 NIL 270 NIL NIL NIL NIL NIL 5 7 CA020400000)
(NIL NIL MT085100010 NIL SP020110000 NIL NIL NIL NIL 1922 R 7 NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL NIL 5 8 CA020400000)

Figure 3. Database records describing problems in a concrete building. These are developed from the transcripts of the site surveys and refer to pre-developed material, structure, symptom, cause and illness codes. The field names are arrayed at the top of the file..

The reported computer based modules built for the architectural firm focus on the second phase of knowledge-based system building; that of eliciting the knowledge and structuring it in a mode compatible with a chosen representation. Development was limited by the pre-existing data structure and data representation.

APPLIED MACHINE INDUCTION TECHNIQUES

The architecture firm's survey methodology produced large amounts of data. About 500 data base records form a hierarchical description of a surveyed building. These records contained a description of the symptoms and signs observed, the context in which they were observed, and the associated causes and illnesses. The assumption underlying the use of the data by machine learning algorithms was completeness of each record and a pathology that included the expertise involved in diagnosing building problems. Machine learning algorithms could then be used to extract this knowledge. The availability and structure of the information suggested the use of a combination of induction techniques, namely learning from examples and learning by observation and discovery.

In the process of learning from examples, a set of examples is provided by a teacher. The example set may or may not contain descriptions of concepts. In one scenario the teacher may generate an example set that is meant to be as helpful as possible; on the other hand, a data base file from a field survey may contain relatively uncontrolled observations. In the present case only positive examples were provided to the learner. The goal of the learner is to classify observations of multiple aspects of the environment, with minimum interaction with the teacher.

Two modes of learning are provided. The first mode is a one-trial method; the induction process has access to all the currently available data base files. The other mode, an incremental approach, closely follows the strategy developed for the rule authoring assistant. The learner must form one or more hypotheses consistent with the available data, and subsequently refine them after considering additional examples.

Comparison of Learning Algorithms; ID3 and AQ11

Two machine learning algorithms were used to construct the rule authoring assistant, specifically a version of Quinlan's Iterative Dichotomizer 3 algorithm (Quinlan 1986) and Michalski's AQ11 program (Michalski 1983). The ID3 variant builds decision trees to discriminate among classes of objects in the domain. The nodes of the decision tree correspond to selected object attributes, such as symptom, material, and orientation, and the edges to predetermined alternative values for these attributes. The leaves of the tree correspond to sets of objects with an identical classification. Decisions as to what attributes are appropriate for branching are made on statistically derived characteristics of the example set such as an entropy type statistic based on information theory or calculation of the chi-squared ratio.

The AQ11 algorithm is typical of the generalization/specialization concepts of machine learning. Generalization begins with the most general possible hypothesis and moves towards more specific descriptions. When a hypothesis fails to cover a positive occurrence it is deemed overly specific and a more general hypothesis is formulated. If a hypothesis covers a negative instance it is overly general and subsequently dismissed. The inverse of generalization is specialization; if a hypothesis is deemed to be overly general (by covering negative instances) it is replaced with a more specific hypothesis. Hypotheses that do not cover all the positive examples are dismissed as overly specific. This type of induction is heuristic in nature using specialization and generalization rules (Michalski 1983) as opposed to strategies such as ID3 which are statistical in nature.

It was possible to compare the performance of both induction algorithms using the data base files provided by the architectural firm. The representation of the data was described earlier. Each algorithm has a distinct representation, ID3 representing knowledge in a classification tree while this particular AQ11 implementation represented knowledge as discrete events and hypotheses using formal logic-based expressions. Both these representations were converted to production rules for use by the architectural firm in their knowledge based system. Classification trees from both ID3 and AQ11 have been derived from the datafile and are shown in Figure 4.

If the complexity of a rule is measured by the number of antecedent clauses and attributes in a rule then ID3 tended to produce more complex rules then AQ11. The rules generated by AQ11 were easier to understand. The explanation for this is the underlying representation of ID3, namely decision trees. The construction of decision trees occurs in a
depth first fashion. Each complex rule generated using id3 included all the previous attributes higher up in the decision tree. When comparing the performance of the AQ11 and ID3 algorithms on identical test data it was apparent that AQ11 was the slower of the two. ID3 seems to demonstrate a linear response with increasing problem size, whereas AQ11 appears to exhibit exponential behavior. Although ID3 is a one step induction algorithm, it was possible to achieve some level of incremental learning with it. This involved considerable user interaction in interpreting results and modifying the subsequent example sets.

A superior feature of AQ11 is the flexibility accorded the expert in incorporating rules of thumb and domain specific information into the induction process. Domain information in the ID3 methodology must be incorporated as attributes, attribute values, and class values. The data file in Figure 3 contains the information of a site survey done on a concrete building. ID3 was used to classify this data file. When the rules were reviewed, the experts commented that there were a few rules that were incorrect or at least incomplete. For certain causes the material attribute was not present, indicating that the types of materials present were not relevant to the illnesses being experienced. This result occurred because ID3 was unable to discriminate between certain class values of illnesses using the attribute "material". Some

```
(CODMAT (MT075100000 (CAUSES (CA020400000))) (MT075400000 (CAUSES (CA010301060)))
 (MT099100010
  (CODSYM
   (SP030200000
    (ORIENT (NIL (CAUSES (CA020400000))) (360 (CAUSES (CA010301060)))
     (90 (CAUSES (CA020400000)))))
   (SP020107000 (CAUSES (CA020400000)))
   (SP040600000
    (ORIENT (360 (CAUSES (CA010300000))) (270 (CAUSES (CA010206000)))
     (90 (CAUSES (CA010206000))) (180 (CAUSES (CA010206000)))))))
 (MT084100010 (CAUSES (CA020400000)))
 (MT033100010
  (STADE
   (5
    (ORIENT (360 (CAUSES (CA010305020))) (270 (CAUSES (CA010305020)))
     (180 (CAUSES (CA030000000)))))
   (3 (CAUSES (CA030000000))) (2 (CAUSES (CA010305020)))))
 (MT092200000 (CAUSES (CA010305020))) (MT085100010 (CAUSES (CA020400000)))
 (MT057000010 (CAUSES (CA020400000))) (MT034100010 (CAUSES (CA010304000)))
 (MT079200000 (CAUSES (CA020400000))) (MT042100000 (CAUSES (CA010304000)))
 (MT042200000 (STADE (9 (CAUSES (CA010305020))) (0 (CAUSES (CA010304000)))))
 (MT081100000 (STADE (5 (CAUSES (CA000000000))) (3 (CAUSES (CA020400000)))))
 (MT057000000 (CAUSES (CA020400000))) (MT079200010 (CAUSES (CA020104000)))
 (MT057100010 (CAUSES (CA020400000))) (MT082100010 (CAUSES (CA020400000)))
 (MT042000000 (CAUSES (CA000000000))))
```

a) - An ID3 decision tree

```
(HYPOTHESES
 ((((STADE = 8) V ((CARACT = 5) (STADE = 9)) V ((CARACT = 5) (CODSYM = SP040600000)) V
   (CODMAT = MT057000010) V ((CODSYM = SP030200000) (ORIENT = 90)) V
   ((ORIENT = NIL) (STADE = 3)) V ((CARACT = 1) (STADE = 0)) V
   ((CODMAT = MT099100010) (CODSYM = SP020107000)) V ((ORIENT = 360) (CODMAT = MT081100000))
   V (CODMAT = MT082100010))
  -> (CAUSES = CA020400000))
 (((CODMAT = MT075400000) V ((CODSYM = SP030200000) (CODMAT = MT099100010) (ORIENT = 360)))
  -> (CAUSES = CA010301060))
 ((((CODMAT = MT033100010) (ORIENT = 270)) V (CODMAT = MT092200000) V (CODSYM = SP010200010)
   V ((CODMAT = MT033100010) (ORIENT = 360)))
  -> (CAUSES = CA010305020))
 ((((CODSYM = SP040600000) (CARACT = 1) (ORIENT = 270)) V
   ((CODSYM = SP040600000) (CARACT = 1) (ORIENT = 90)) V ((CARACT = 1) (ORIENT = 180)))
  -> (CAUSES = CA010206000))
 (((CODMAT = MT034100010) V (CODSYM = SP060100000)) -> (CAUSES = CA010304000))
 ((((CODMAT = MT033100010) (STADE = 3)) V ((CODMAT = MT033100010) (ORIENT = 180) (STADE = 5)))
  -> (CAUSES = CA030000000))
 ((((CODMAT = MT081100000) (STADE = 5)) V (CODMAT = MT042000000)) -> (CAUSES = CA000000000))
 (((CODSYM = SP040600000) (CARACT = 1) (ORIENT = 360)) -> (CAUSES = CA010300000))
 ((CODMAT = MT079200010) -> (CAUSES = CA020104000))))

(END)
```

b) - an AQ11 rule

Figure 4. A comparison of the generated classifications for the same data using ID3 and AQ11.

domain knowledge, such as; materials and illnesses must be related,, or some domain facts, such as; the building is made of concrete, would have to be included to ensure that rules were generated in the proper context.

The information contained in building surveys can be numerical as well as categorical. Information such as material quantities, percentage of areas affected and ages of building and materials with a symptom are recorded. The ID3 algorithm was extended to allow for the inclusion of numerical attributes. Numerical attributes are converted into partitions, which are represented as Boolean attributes. One method of partitioning a numerical attribute space is to break the space at the midpoints. For example, given the following data;

building age = 10, 20, 40;

the space would be partitioned into the following Boolean attributes;

building age < 30, building age < 15;

Several other partitioning heuristics, such as partitioning around the mean, were also provided, as well the ability to include other heuristics as needed. AQ11 treats numbers as categorical values and generated some hypotheses that were difficult to follow.

A difficulty with both methods was missing information. The data supplied by the architectural firm consisted a of relatively square data base, having approximately 30 attributes for 60 record entries. Approximately half of the records were empty. That fact led to the production of rules where the lack of an attribute value sometimes became a relevant factor. A portion of a rule generated is shown in Figure 5. Over one half of the attributes' values are NIL. The rule contains clauses with up to 30 attributes. However, when the attributes containing no information are excluded from the induction process the rules become more understandable. In fact after several trial runs with this data base it was found that only 3 out of 30 attributes, symptom, material, and orientation, proved necessary to classify the building illnesses.

Manual Editing of Rule-Bases

The experts wished to include additional levels of attributes in the generated rules, in order to arrive at descriptions of illnesses that were meaningful. This could have been achieved by providing negative or counter- examples in the data, forcing the induction algorithms to include certain parameters; however, it was found easier to add antecedent clauses to specific rules. In the example generated from the concrete building, the additional attribute "MATERIAUX=BETON" was added to some of the rule clauses.

Allowing the rule-authors to change and develop their own rules introduces problems:

1. the rules may no longer be logically consistent
2. the edited rules no longer precisely reflect the data recorded on site
3. the semantic consistency may change with addition of new attributes (to define meta-level causes and diagnoses)

Only the first of these concerns was effectively dealt with. A rule verification algorithm was developed to maintain logical consistency in the rule base.

```
(HYPOTHESES
 (((((CODZON = NIL) (CODSYS = NIL) (CODMAT = MT099100010) (SYSMAT = NIL) (CODSYM
= SP040600000)
   (EMPLAC = NIL) (PIECE = NIL) (ETAGE1 = NIL) (ETAGE2 = NIL) (AGEMAT = 1967)
(INTMAT = A)
   (ENTMAT = 5) (USAMAT = NIL) (QUAMAT = NIL) (UNIMAT = NIL) (AGESYS = NIL)
(INTSYS = NIL)
   (ENTSYS = NIL) (USASYS = NIL) (MILIEU = 5) (EXPOSI = 9) (FACADE = NIL) (ORIENT =
270)
   (SUPERF = NIL) (VALEUR = NIL) (AFFECT = NIL) (QUASYM = NIL) (UNISYM = NIL)
(CARACT = 1)
   (STADE = 3))
  V
  ((CODZON = NIL) (CODSYS = NIL) (CODMAT = MT099100010) (SYSMAT = NIL) (CODSYM
= SP040600000)
   (EMPLAC = NIL) (PIECE = NIL) (ETAGE1 = NIL) (ETAGE2 = NIL) (AGEMAT = 1967)
(INTMAT = A)
   (ENTMAT = 5) (USAMAT = NIL) (QUAMAT = NIL) (UNIMAT = NIL) (AGESYS = NIL)
(INTSYS = NIL)
   (ENTSYS = NIL) (USASYS = NIL) (MILIEU = NIL) (EXPOSI = 9) (FACADE = NIL) (ORIENT =
90)
   (SUPERF = NIL) (VALEUR = NIL) (AFFECT = 2.0) (QUASYM = NIL) (UNISYM = NIL)
(CARACT = 1)
   (STADE = 3))
  V
  ((CODZON...
```

Figure 5. Internal rule syntax used by the modules and editor before translation to a specific KBS shell format.

KEEPING THE RECORD STRAIGHT: Rule-base Verification

The automatic analysis of a data base by means of an induction algorithm will not, of course, give rise to a logically inconsistent rule set. However when the rules produced by multiple applications of the algorithm, using different combinations of parameters, are merged then the resulting rule-base may contain examples of redundancy and/or subsummation. As rules derived from additional data-bases are added to the knowledge-base, and rules are added or modified manually by the experts, the danger of compromising the logical integrity of the system increases.

In order to maintain the logical consistency of the rule-base, it was necessary to adopt explicit verification methodologies, and implement them in such a way that they could be used during development, after each modification or addition. Verification is a process supplementary to validation, which tests the performance of a system and can therefore only be used when the system, or a prototype, is largely complete. Validation techniques treat the system as a "black box", comparing the conclusions arrived at by the system with those of the expert given the same initial data. When there is failure to correlate, adjustments are made to the system. Generally, a system is developed with reference to an example set of data and solutions, extended to cope with extreme cases, and tested on new data. Although it is an essential part of acceptance testing, this process cannot constitute a formal proof since it is inductive; however comprehensive the testing there is no guarantee that further, unconsidered, circumstances will not prove disastrous. Moreover, confidence in the reliability of a system requires a more solid base than assurances that it has worked so far. Not only must the system be correct, it must

be seen to be correct. To achieve this level of confidence, it is necessary to verify the internal logic of the knowledge-base and inference engine (Suwa *et al,* 1985).

In a rule-based system there are several possible causes of error or less than optimal performance:

1. Rules may be in conflict,
2. Rules may be redundant.
3. Inference chains may be circular.
4. Rules, inference chains or attributes may be isolated, either never invoked by the rest of the system or not able to be fulfilled.
5. Attributes may be incomplete, having some range of their value unreachable (Nguyen *et al,* 1985).

Where the rule-base is small and developed by one person over a limited period of time, such faults can usually be discovered fairly readily by inspection. However in more complex situations (the construction industry is a prime example), the process of assessing logical integrity is non-trivial. Problems arise when the knowledge is fragmented, and contributions from several experts are to be incorporated. Experts may suggest rules which contradict, conflict with or overlap rules from other experts; they may use different terms for the same concept or the same term for different concepts, and the root of these differences may be conceptual or merely syntactic. Such problems may also occur when new knowledge is added to an existing rule-base, particularly where the additions are not made by the original knowledge engineer.

In these situations an automatic verification system is useful, not only to increase the level of confidence in the completed rule-base but as a tool during development. Since a knowledge-base is essentially a logical system, it is amenable to logic-based verification techniques (Stachowitz, Coombs and Chang, 1987).

Rule-base Logic

In order to evaluate the logical dependencies in a rule-base, it is necessary first to examine the structure of a single rule. The antecedent of the rule may consist of one or more "literals", simple assertions of the type "attribute A has value B" or "A = B". These literals may be linked conjunctively or disjunctively, i.e. by the Booleans "and" or "or". Conjunctive literals are each necessary but not sufficient for the rule to fire: disjunctive literals are sufficient but not necessary. A disjunctive clause may consist of one or more conjunctive literals. It follows that a literal which is not conjoined with another literal is either a complete or at least a sufficient antecedent for the rule. When comparing rules, it is necessary to compare not only complete antecedents but also sufficient antecedent clauses. The consequent clause of the rule is generally a single literal; rules with multiple consequents may be considered as a set of separate rules with identical antecedents.

The following types of logical inconsistency in a rule-base have been identified:

(1) Conflict
When the same antecedent (or sufficient antecedent clause) occurs in two rules whose consequents are contradictory, there is obviously a problem - one rule at least is incorrect:

if A is B **or** C is D
then P is Q

if A is B
then P is not Q

Where the consequents reach different conclusions about the same attribute, the issue is context-dependent:

if sky is cloudy
then temperature is low

if sky is cloudy
then temperature is high

represents a conflict, whereas

if sky is cloudy
then temperature is low

if sky is cloudy
then temperature is likely to change

does not.

Rarely, a single rule may be self-contradictory:

if A is B **or** C is D **or** E is F
then C is not D

(2) Redundancy
There are several ways in which rules can overlap. While this is not usually a fatal error, it can lead to inefficiency, undue weight being given to repeated information, or can mask mistaken omissions. A rule may be redundant:

if A is B **or** C is D
then P is Q

if C is D **or** A is B
then P is Q

Where two rules reach the same conclusion, and have equivalent antecedents or antecedent clauses, then either one rule is subsumed by the other, or parts of them are redundant.

if A is B **or** C is D
then P is Q

if A is B
then P is Q

first rule subsumes second

if A is B
then P is Q

if A is B **and** C is D
then P is Q

first rule subsumes second

if A is B **or** C is D **and** E is F
then P is Q

if C is D **and** E is F **or** G is H
then P is Q

one occurrence of the condition "C is D and E is F" is redundant.
Note that:

if A is B **and** C is D
then P is Q

if C is D **and** E is F
then P is Q

is not an example of redundancy; although it would be possible to combine the rules, the resulting disjunction would be logically equivalent to the two separate rules.
Where the antecedents are almost equivalent but contain contradictory literals, and yet reach the same conclusion, that literal is an irrelevant condition:

if A is B **and** C is D **and** E is F
then P is Q

if A is B **and** C is D **and** E is not F
then P is Q

"P is Q" does not depend on the truth of "E is F".

(3) Circularity
When a rule refers to itself, either directly or indirectly, then on firing that rule the system will hang.

if A **and** B **or** C
then B — self-reference

if A **and** B **or** C
then P

if P **or** Q
then X

if X
then C — circular inference chain.

(4) Rules whose conclusions are not accessed by other rules, and do not constitute goal-states, represent "dead-ends" in an inference chain. Rules with antecedent conditions which are not able to be fulfilled (from initial or user-interface input, or as a consequent of another rule), cannot fire. Incompleteness of chains is flagged.

(5) An attribute may be incomplete, having some part of its allowable range of values not covered by any rule. This check was not implemented in the verification module.

CIRCE

The verification system CIRCE (Conflict, Incompleteness, Redundancy and Circularity Evaluator) was developed as a knowledge acquisition tool for the building diagnosis package. It was written originally on a Lisp machine and then ported to an AT class microcomputer running Common Lisp, for integration with the classification module. CIRCE examines the rule-base, checking each rule for internal problems, then comparing it with every other rule in turn and noting interaction problems and dependencies. Finally, each inference path is checked for faults.

The system first sets up empty stacks for the list of rules and the reports of faults. It then calls, serially, five sub-modules performing the necessary procedures of reading in the rule-file, separating out the logical components of the rules, comparing the rules and charting the logical dependencies, investigating inference chains, and reporting the results.

Reading the Rule-file The first sub-module opens the specified file and passes on each rule in turn, in the form of a list. The first item in each list (which is the name of the rule) is added to the global variable "rulestack" and the name and text passed on. The data-structure used for the rules is the "frame" (Winston & Horn 1984). Frames can have an unlimited number of slots, each of which can have an unlimited number of facets. A value is associated with each facet. A frame is set up for each rule, referenced by the rule-name. The antecedent and consequent portions of the text are separated and placed on the frame.

Parsing the Rules The parsing submodule analyses the rules ready for logical comparisons. First, in cases where the consequent contains a negative, 'not' is placed at the beginning of the text. Next, the text of the antecedent is divided, by splitting on "or", into separate disjunctive parts; these are further divided, by splitting on "and", into separate conjunctive clauses, which are recombined after alphabetical sorting. Each clause is given a unique reference.

Note that a disjunctive clause may contain one or more conjunctive clauses, and is always a sufficient condition for the rule. An antecedent may contain one or more disjunctive clauses. A rule whose antecedent consists of multiple disjunctive clauses is logically equivalent to that number of separate rules, each with the same consequent. A conjunctive clause is always a single statement, a 'literal', but may also be disjunctive, i.e. constitute a complete antecedent.

Checking Logical Dependencies The next sub-module checks the logical relationships between rules. Each rule in turn is first examined for internal problems, (self-contradiction and self-reference), then compared with every other rule on the remainder of the list of rules. In this way, each rule is checked only against those rules with which it has not yet been compared.

Rules with equivalent antecedents or antecedent clauses are checked for several possible errors. If the rule consequents are contradictory (one is the Boolean complement of the other); there is contradiction; if they refer to the same parameter but reach different conclusions there may be conflict, depending on the type of parameter. As no information about parameters is available in this implementation, the situation is simply presented to the user and the parameter and the rules concerned are posted to the appropriate variable.

If the rule consequents are the same, there is repetition, which can lead to inefficiencies and inaccuracy in the rule-base as the same information fires two rules. When the rules are identical, the situation is reported as redundancy; when there are alternative antecedents the less constrained rule is described as subsuming the more constrained (i.e. the rule with fewer alternatives). If the consequents are the same and the rules have contradictory antecedents, or

complex antecedents that are the same except for a single literal asserted in one and denied in the other, that condition is irrelevant to the truth of the consequent.

Finally, information on the dependencies between the rules is compiled in preparation for the next stage.

Examining Inference Paths The information recorded while searching the rule-base is now used to examine all possible inference paths. Starting from each rule in turn, those rules for which the current rule's consequent is a condition are checked - if the consequent of the rule examined is a literal in the antecedent of the original rule, there is a circularity. The descendants of each of the rules are also checked, enabling circularity to be detected however long the inference chain. The search is depth-first.

Reporting When the verification procedures are completed, if no faults have been reported a message is sent to the screen. If there are faults, the appropriate report-printing function is called with the material recorded for each instance of the fault. A file is opened for the results, and for each fault appropriate explanatory text and the text of the rules is printed.

Use of CIRCE

After editing, the verification procedure should be re-called until no faults are reported. Search time depends on the number of comparisons to be made, which is less than (number of rules) squared. Comparisons take minimal time if there are no similarities between the rules, so the total time needed depends on the complexity of the rule-base; a rule-base of fewer than 50 rules, designed to incorporate an example of every fault-type described above, was processed in under 3 minutes on an AT class machine. This short processing time should encourage regular use of the verification procedure, whenever modifications of any substance have been made to the rule-base.

DIAGNOSIS

The system has been used on relatively small files (60 records) due to problems in the semantics and syntax of the symptom, cause and illness descriptions. The rules generated by the induction algorithms were perceived by the experts to be "obvious" and "trivial". On the other hand, even with the small data sample, the architects discovered relationships and patterns they had not considered before.

Verification of other diagnostic rule bases, such as those found in a previous rule base for diagnosing problems with windows, found 2 redundancies among the 195 rules and brought 18 potential conflicts to the author's attention, 3 of which were other than allowable multiple values. In all 5 faults were identified.

This system was envisioned as underpinning a long-term effort by the firm to develop the knowledge base. The tool has actually pointed to weakness in the representation of building faults.

Representation During the data representation development phase the firm had assumed experts were using data in the records to arrive at diagnoses when in practice they were referring to the natural language problem descriptions found in the memo fields. In the test rule file which excluded memo fields, it was found the expert generally conjunctively linked about 3 or 4 records together to arrive at a specific diagnosis. A general diagnosis involving many building systems (such as walls or structure) would require a meta-level diagnostic descriptor.

Although the development of the knowledge acquisition system hinged on the completeness of problem descriptions found in each record, both the symptom and cause fields in the database were often missing. During the development phase, the necessity of record independence and completeness was stressed. Because of the lack of completeness of information in each record, patterns derived from the records proved insufficient to generate meaningful rules. Six problems with the database were identified:

1. the records comprised alphanumeric codes. These had to be translated into meaningful sentences each time the records were accessed and assessed,
2. records were incomplete; data was insufficient to arrive at a cause for the symptom being recorded on each record,
3. the natural language descriptors were found to be the key information sources for experts performing diagnoses. These memo fields had no syntactic formality (except that naturally ascribed to the French language).
4. without a memo field, a relevant cause could be identified only if many records were linked together (conjunctively),
5. the fields were insufficient to describe some problems (e.g. the above window problem) and too detailed for describing others (building cracks). The "grain size" was invariable yet the problem descriptions required variability,
6. there was no mechanism for coding spatial information.

Shortcomings of the data structure have been referred to above. However, without the knowledge acquisition and verification tool, these shortcomings would not have been made explicit and confronted. Solving the representational issues would not have been attempted.

Learning Algorithms Experimentation with the two learning programs described above led to the inclusion of one of them into the system developed for the architectural firm. The ID3 variant was chosen. The structure of the data base information was particularly well suited to the representation used by ID3. Attributes and attribute values could be generated from the data base file headers. ID3 was also considerably faster than AQ11. Although overnight computer runs are acceptable for academic environments the constraints of a small to medium size architectural firm do not allow the luxury of parametric analysis with slow computer programs. The ID3 algorithm is also conceptually easier than AQ11 to understand and modify.

The information generated by the machine induction algorithms did not produce the anticipated results; general rules about building diagnostics. There were several reasons for this. The database size was over-stated. It was originally anticipated that files containing several hundred records would be used. The largest file tried to date contained 400 records; however, the file was not provided by the architectural firm, and contained numerical and categorical data about air infiltration rates. The size of the files did not allow ID3 or AQ11 to discover any rules of consequence. In fact the rules generated were extremely focussed and applied only to the particular data set from which they were generated. The data appeared to be too fine grained to adequately represent the problems occurring in the building. This situation could be improved by recognizing the linkages between various records in that data base file and allowing attributes to have multiple values.

In some cases however over-generalization was a problem. This occurred when information, such as the concrete example mentioned above, is intended to be background knowledge but is included in the data or is absent. In cases such as this over-generalized rules were produced. For example the rule;

IF system = exterior wall AND orientation = 360
THEN cause = spalling

was generated. If the material happens to be concrete then the rule holds; however, if the material is wood then the rule is nonsensical. To present this background knowledge in the data demands more examples or counter-examples. The induction algorithm did find a pattern relating cracks in masonry walls to orientation and solar exposure. This finding had not been considered by the expert previously.

Verification The verification algorithms address a subset of the problems encountered when manual "ad hoc" rule generation is allowed. Redundancies, conflicts and incompleteness arising between pairs of rules only are considered here; except when searching for circular inference chains, CIRCE does not look at more complex relationships. It is therefore possible that some inconsistencies and redundancies arising between larger combinations of rules may be overlooked (Ginsberg, 1987). It was however felt that the very significant increase in processing time and computational resources required to perform a complete investigation made such a system inappropriate for this application, more especially since the rule-base although potentially extensive is at this stage shallow. When the knowledge base is more fully developed, multi-level verification procedures and an optimisation capability should be implemented.

CIRCE brings logical inconsistencies to the user's attention, but does not correct them as it does not have access to the user's intentions. Determination of the logical relationships between rules is a question of syntax, but semantics may influence the appropriate modifications; Eg. where two rules reach different conclusions about a single parameter, based on the same evidence - the rules may be complementary or in conflict, depending on the context. Automatic elimination of irrelevancies and redundancies would be possible; however, this may be counter-productive in some instances, e.g. where an incorrect attribute name was used. It is felt that simple notification is the safest option.

Without semantic information, the truth of a rule cannot be automatically determined; it is quite possible for a system to be logically impeccable while not modelling a process in any useful way. Verification does not constitute a proof that a particular rule-based system is the best or even an acceptable solution. It is however a necessary (though not sufficient) step in the process of developing a working knowledge-based system.

PROGNOSIS

The knowledge acquisition and verification system has pointed to:

- a need for a universal taxonomy and pathology for buildings. Experts in the building industry must first come to agreement about the syntax of the problems they describe before a production system can be built addressing the problem.
- a need to better understand what spatial information is required to describe the pathology.
- a need to define a verification system that has partial semantic checking as well as the syntactic checking provided by CIRCE.

Before knowledge acquisition can be properly undertaken in the development of a system, the first phase, knowledge representation, should be completed. In order to reach an informed decision about the most appropriate representation, it is essential to develop familiarity with the terms and concepts used in the domain. In practical terms this means that some initial knowledge acquisition work, probably informally structured, should precede detailed consideration of the representation.

The representation of diagnostic knowledge in this domain is plausible using production rules,

but requires underlying system models for acceptance. A backward chaining mechanism would be the natural inference mechanism to use during analysis of problems but the data structure imposed by the client in this case implied a forward inference mechanism. The choice of inference mechanisms was less of a concern than the development of a system for authoring and maintaining knowledge derived from a database that did not capture the required information.

Although no rules of any consequence were generated for the architectural firm, the application of these techniques provided useful insights into the representation of building problems. It particularly identified the need for multiple record linking, meta-level diagnostic knowledge and geometric representations. By analyzing the data they were able to determine for the first time the usefulness of what they had been collecting and to explicitly map portions of their procedure that worked and techniques that did not. For example, the memo fields were extremely valuable, but the zone description hierarchy was poorly conceived to represent spatial information.

The long term use of this rule authoring method will produce an orderly and consistent knowledge-base. The module and an associated inference engine (or knowledge-based system shell) fit into the firm's existing method for diagnosis. The procedure produces an explicit record of problems. Developing new representations that better describe the building faults has begun:

- Geometric information is being coded on a generic building type to provide the expert with a model for the spatial continuity of symptoms and connection of building systems.
- Representation of failure-prone building components by fine-grained objects may provide the required variation in information grain size.

The issue of taxonomy and pathology must be addressed by the building industry practitioners as a whole. Eventually it is hoped this will lead to complete pathologies and representations supporting not only diagnosis but also design.

ACKNOWLEDGEMENT

We gratefully acknowledge Patrice Lapointe and his firm COPLANAM Ltee's contribution to this work. Without their knowledge of the problems, this work would not have been possible.

REFERENCES

Bayazit, N. (1984). A Study on the Diagnosis of Building Use Failure, *Design Methods and Theories*, (19), 2,Design Methods Group, San Luis Obispo, California. pp.268-280.

Boose, J.H. (1986). *Expertise Transfer for Expert System Design*, Vol.3. New York, Elsevier.

Davidson, C.H., Davidson, P.L., Ruberg, K. (1988). "Expert Systems and the Use of Information in Building Design and Construction", *The Jrnl of Documentation*, (44), 2, June 1988, pp. 91-118.

Ginsberg, A. (1987). A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy. In *Proceedings of the Third Annual Expert Systems in Government Conference*, Washington D.C. 1987.

Hoffman, R.R.,(1987). The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology", *AI Magazine*,(8),2, American Association of Artificial Intelligence, Live Oak Press. pp.65-67.

Kahn, G.S., Kepner, A., Pepper, J. (1987) TEST: A Model-Driven Application Shell, *Proceedings of the AAAI-87, Sixth NAtional Conference on Artificial Intelligence*, Seattle, Wa. July 13-17, 1987, pp. 814-818.

Michalski, R.S. (1983) A Theory and Methodology of Inductive Learning, *Machine Learning: An Artificial Intelligence Approach*, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, (eds.), Tioga Publishers, Palo Alto, Ca.lifornia, pp. 83-134.

Michalski, R.S., Mozetic, I., Hong, J.; Lavrac, N. (1986), The Multi-Purpose Incremental Learning System AQ15 and its Testing Applications to Three Medical Domains, *Proceedings AAAI-86, Fifth National Conference on Artificial INtelligence*, August 11-15, 1986 Philadelphia, Pa., Morgan Kaufman Publishers, California. pp.1041-1044.

Nguyen, T.A., Perkins, W.A., Laffey, T.J. and Pecora, D. (1985). Checking an Expert System's Knowledge Base for Consistency and Completeness. In **Procedings of the Ninth International Joint Conference on Artificial Intelligence**, 374-378. Menlo Park,CA: American Association for Artificial Intelligence.

Pazzani, M.J. (1986) Refining the Knowledge Base of a Diagnostic Expert System: An Application of Failure-Driven Learning, *Proceedings AAAI-86, Fifth National Conference on Artificial INtelligence*, August 11-15, 1986. 1029-1033. Philadelphia, Pa., California: Morgan Kaufman .

Quinlan, J.R. (1986) Induction of Decision Trees, *Machine Learning*,(1), 1,81-106. Boston, Mass: Kluwer Academic Publishers.

Shortliffe, E.H. (1976) *Computer-Based Medical Consultations:MYCIN*, New York;
Elsevier

Stachowitz, R.A., Combs, J.B., and Chang, C.L., (1987) Validation of Knowledge-Based Systems, *Proceedings Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program*, Arlington, Virginia. March 9-11, 1987.

Suwa,M., Scott, A.C. and Shortliffe. E.H. (1982). An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System. *AI Magazine* 3(4): 16-21.

Tanaka, T. (1986) Parsing Circuit Topology in a Deductive System, *Proceedings AAAI-86, Fifth National Conference on Artificial Intelligence*, August 11-15, 1986 Philadelphia, Pa.,1029-1033 CA; Morgan Kaufman.

Waterman, D.A. The Rand Corporation (1986), *A Guide to Expert Systems*, 272-289. Reading MA; Addison Wesley.

Welbank, M. (1983), *A Review of Knowledge Acquisition Techniques for Expert Systems*, 16-49. Martlesham Consultancy Services, British Telecom Research Laboratories, Martlesham Heath, Ipswich, England.

Wilson, F. (1984), *Building Materials Evaluation Handbook*, 358. New York; Van Nostrand Reinhold.

Winston, P.H. and Horn, B.K.P. (1984). *Lisp*. Reading, Mass; Addison-Wesley.

Xiang Z., Srihari, S.N., Shapiro, S.C. and Chutkow, (1984) Analogical and Propositional Representations of Structure in Neurological Diagnosis, *Proceedings of the First Conference on Artificial Intelligence Applications*, IEEE Computer Society, December 1984.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Using Personal Construct Techniques for Collaborative Evaluation

Douglas Schuler, Peter Russo, John Boose, and Jeffrey Bradshaw
douglas@Boeing.com
Knowledge Systems Laboratory
Advanced Technology Center
Boeing Computer Services
P.O. Box 24346, M/S 7L-64
Seattle, Washington 98124 USA

Abstract

Efforts are underway to better characterize group processes at The Boeing Company as part of a project to design software for computer supported collaboration. This paper describes work in progress to support multi-user, collaborative situations using Aquinas, a knowledge acquisition workbench. An experiment is described in which Aquinas is used to facilitate the collaborative evaluation of an in-house Boeing Advanced Technology Center course in knowledge engineering.

Introduction

This paper describes how Aquinas, a workbench that supports the knowledge acquisition phase of knowledge engineering, was used to improve the evaluation of courses at the Advanced Technology Center of Boeing Computer Services (ATC). Aquinas uses techniques from personal construct theory (Kelly, 1955) to elicit and analyze student-generated course evaluation criteria. These criteria are represented in the form of "grids". Individual grids were combined using Aquinas in order to arrive at group evaluations. The process of deriving group opinions through individual and collaborative effort can reveal different information than would be obtained through a traditional evaluation approach. The initial phases of this process in many ways resemble a Delphi approach to consensus eliciting (Helmer, 1966) but later phases take advantage of focused group discussions.

This section describes the problem that our work addresses, the context of the work and the tool that we used. The Approach section describes the overall approach that we took and the rationale that we used. The Collaborative Evaluation section describes the purpose and goals of the experiment, and the specific procedure that we followed. The Process Evaluation section evaluates the effectiveness of the process from various viewpoints, including the instructor's, the students', and the researchers'. Finally, the Discussion section discusses other relevant considerations and opportunities which we plan to pursue.

Problem

Student feedback on classes is crucial to technology transfer within The Boeing Company. This is usually done through multiple-choice evaluation sheets which are filled out by the students at the end of each class. The information on the evaluation sheets is frequently augmented through discussions with participants during and after the classes. The evaluation sheets are used in an ad hoc manner. There is no formal method for assessing group opinions or for comparing results from different classes. There are always aspects of the class which are not mentioned on an

evaluation sheet and students may have needs arising from their job which are not obvious to the instructors. The result of this is that sometimes the most important criteria relevant to the evaluation of a course may be missed completely by a particular form. We set out to develop an approach to gathering and analyzing evaluation data that would address these problems.

Objectives

One of the most important roles of the ATC is to disseminate knowledge and offer assistance in the development and use of advanced computing technology throughout the rest of The Boeing Company. Organizations in the company may sponsor "AI associates" to come to the ATC for training and to develop a prototype of an application useful to their home organization. ATC researchers advise associates and teach classes on such subjects as knowledge-based systems, natural language processing, and high-speed computing. The ATC also contains a knowledge systems laboratory and a high-speed computing laboratory.

In the teaching of courses on artificial intelligence (AI) principles and development techniques, classroom time is limited. Hence the value of assuring that course content is focused on the most current and useful topics. To most students, the Knowledge Engineering class embodies the culmination of AI training, since it integrates previous coursework on theoretical underpinnings with the real, pragmatic concerns of their particular application. Improving the course evaluation process for the Knowledge Engineering class was seen as both a needed objective and a suitable proving ground for our collaborative support objectives.

We had five objectives in our work on collaborative evaluation. Although these objectives could not be completely met as the result of a single experiment, we wanted to keep the context of the work foremost in our thinking. The objectives included (1) assisting the instructor in evaluating specific courses; (2) defining a set of collaborative work processes; (3) extending Aquinas to address this set of features; (4) assessing the effectiveness of the collaborative processes; (5) assessing the effectiveness of using Aquinas to implement these processes. Although all objectives will be considered, success of the approach will be ascertained primarily through consideration of (1) above — that is, did the collaborative process help the instructor to evaluate the course?

Aquinas - A Knowledge Acquisition Workbench

Knowledge acquisition is the formulation of expert knowledge into a representation that can be used by a knowledge-based program. It is seen as an early (and ongoing) phase of knowledge engineering. Aquinas, an expanded version of the Expertise Transfer System (Boose, 1984; Boose, 1986b), is a workbench that supports several knowledge acquisition activities. The workbench combines techniques from psychology and knowledge engineering. Its activities include eliciting distinctions, decomposing problems, combining uncertain information, incrementally testing knowledge bases, integrating data types, automatically expanding and refining the knowledge base, using multiple sources of knowledge, and providing guidance about the process itself. Aquinas interviews experts and helps them analyze, test, and refine their knowledge base. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined. Results from user consultations are derived from information propagated through hierarchies. Distinctions captured in grids can be converted to other representations such as production rules, fuzzy sets, or networks. Aquinas delivers knowledge by creating knowledge bases for several different expert system shells. Help is given to the expert by a dialog manager that embodies knowledge acquisition heuristics (Kitto, 1987). Aquinas is written in Interlisp and runs on the Xerox family of Lisp machines. Subsets of Aquinas also run in an Interlisp version on the DEC Vax and on the Sun family of workstations. A portable C-based version has been ported to the Macintosh (Bradshaw, Boose, Covington, & Russo, 1988) and to a variety of UNIX platforms. Aquinas is discussed in greater detail in Boose and Bradshaw (1987), Boose, Shema, and Bradshaw (1988), and Boose (1988).

Using Aquinas, small rapid prototypes of a knowledge-based system can be built in as little as one hour, even when the expert has little understanding of knowledge-based systems or has no prior training in the use of the tool. Aquinas uses methods from personal construct theory, an approach that grew out of George Kelly's research and experience as a clinical psychologist (Kelly, 1955). Kelly's methods and theory provide a rich framework for modeling the qualitative and quantitative distinctions inherent in an expert's problem-solving knowledge. Aquinas can be used to gather and combine information from individuals in support of group decisions and other collaborative processes. It is this aspect of Aquinas that is emphasized in this experiment.

Aquinas represents problem-solving knowledge in hierarchical networks that can be displayed as repertory grids (also called rating grids). A repertory grid is essentially a two-dimensional array. Solutions or entities are aligned along one dimension and traits or attributes are aligned along the other. Traits represent dimensions that may be defined with nominal, ordinal, integer, or ratio-scaled values. Repertory grids hold ratings; each rating is a judgment by the expert relating a solution to a problem-solving trait. In a repertory grid, problem solutions are elicited and placed across the grid as column labels, and traits of these solutions are listed alongside the rows of the grid (Figure 1). Traits are elicited first by presenting groups of solutions and asking the user to discriminate among them (triadic elicitation). Then, the user gives each solution a rating with respect to the trait (for example where the solution "ice cream" falls with respect to "cold" and "hot" ends of "temperature" scale). The user then assigns a relative importance to each trait. After an initial grid is constructed, Aquinas helps the user refine and expand the knowledge base by invoking a variety of analysis tools. Similarities between solutions and traits are analyzed to help the user refine useful distinctions and eliminate those that are inconsequential or redundant. Inductive implications between trait values can show the user higher levels of abstraction implied by a rating grid. If the user disagrees with an implication, Aquinas helps the expert refine the grid.

COURSE CONTENT

Create traits and make ratings in terms of success or contributions to the indicated course components. Use a scale of 1 - 7 to rate each course component. Use the circled items to create triads with which to establish traits. Be sure to include trait weights (from 1 to 7) in the third column.

1. Knowledge Representation	2. NL and other user interfaces	3. Uncertainty	4. System Design	5. Knowledge Acquisition	6. AI Tools	7. Technology Transfer	8. Managing AI software projects	9. Explanation	Trait (1)	Opposite (7)	Weight
2	2	2	2	6	2	6	6	2	Applications	Background Training	4
6	6	6	6	6	6	2	6	6	E/S Development	E/S Use	4
6	6	6	2	4	6	2	2	6	Available Capabilities	Use of Capabilities	4
6	6	6	2	2	6	2	2	6	Background Development	E/S MFGR	4
6	6	6	6	6	6	2	6	4	E/S Methods	Goal	4
2	2	2	6	6	2	4	6	4	Application Driven	E/S Tool Driven	4

Figure 1.

Aquinas represents knowledge in repertory grids. In this case, elements of the knowledge engineering class are listed vertically across the top of the grid. The student-generated traits are listed down the right side of the grid. The students rate each element along the scale defined by the trait and its opposite, and assign a relative importance (weight) to each trait. Each student used a similar sheet was also used to generate individual traits by triadic elicitation (see above).

Rating grids in Aquinas can be combined and arranged hierarchically. This knowledge can be used to solve specific problems by running consultations. During a consultation, Aquinas asks the user to specify evidence and preferences with respect to traits for a given set of solution candidates. Consultation results are displayed as a ranked-list of solutions. Accompanying each solution is a

set of numbers representing the strength of recommendation from knowledge sources, jointly and individually.

Knowledge from multiple experts may be rapidly combined using Aquinas. Users may receive dissenting as well as consensus opinions from groups of experts, thus getting a full range of possible solutions. Disagreement between the consensus and the dissenting opinion can be measured to derive a degree of conflict for a particular consultation. Aquinas has been used for cost- effective group data gathering (Boose, 1986a).

Approach

The course evaluation process was conducted in the following manner. The instructor identified important aspects of class content and class format. Each student used those aspects to identify traits that would be used to evaluate the class. Students then filled out grids consisting of solutions paired with the traits. These grids were then analyzed individually using Aquinas. Discussions between rating sessions were held to orient the students, clarify ambiguous traits, remove unnecessary or duplicate traits, and add additional traits if necessary. The traits that remained were combined into a larger grid which was rated individually by all the students. The larger grid was analyzed using Aquinas, and the instructor examined and interpreted the output. Finally, the entire process was examined formally and informally for effectiveness. Repertory grids were the primary data objects used to collect and analyze data used in all phases of this approach. It should be noted that we tested a collaborative process consisting of many coordinated tasks. This makes final analysis as to effectiveness more difficult but more accurately reflects real-world collaborative demands.

The guidelines that we established make the process unique. These guidelines include keeping data independence as long as possible, allowing students to create their own evaluation criteria for a set of course attributes, using individually gathered data to support group discussion, and analyzing patterns and results after combining the data.

Collaborative Evaluation

Purpose

Course evaluations usually are isolated activities, conducted in such a way so as to preserve anonymity. There is little student input into the choice of either the topics of the evaluation or the criteria by which these topics are to be evaluated. The opportunity to differentiate among the topics and criteria and to rank them by their importance or value is also lacking. Nor do we tell the students which of their inputs were useful or what their fellow classmates thought. The failure of evaluations in these respects reduce their usefulness. Feedback to instructors is especially useful when students represent varied and sophisticated viewpoints, which is true in this case. We proposed a collaborative process to specifically addresses these failings.

The goals of this experiment were to minimize the isolation of the evaluation process, to maximize student input by allowing them to individually identify relevant criteria for assessing the value of course material, and to give the students the opportunity to discuss and criticize these criteria in terms of their relative weight and preferences. Another goal was to furnish the instructor with richer, more useful class evaluations.

Procedure

The course evaluation covered two areas: content and format. Topics for evaluating both were supplied by the instructor. The content topics were: (1) knowledge representation, (2) natural

language and other user interfaces, (3) uncertainty, (4) system design, (5) knowledge acquisition, (6) AI tools, (7) technology transfer, (8) managing AI software projects, and (9) explanation. The format topics were: (1) guest lectures, (2) notebook reading, (3) student seminars, (4) student presentations, (5) textbook reading, (6) knowledge acquisition phase of team project, (7) design phase of team project, (8) prototyping phase of team project, and (9) documentation and presentation phase of team project.

Each student was given a grid form for content and asked to supply six traits through triadic elicitation (Figure 1). Students could also supply additional traits if they wished. Thus, all traits used in the evaluation were generated by the students themselves. Each topic was then rated against pairs of traits on a scale of 1 - 7 (where 7 = most important and 1 = least important). This process was repeated for the course format. This is illustrated in Figure 1. The data was recorded in Aquinas and analyzed using several tools. Seven students participated in this phase.

The trait pairings were collected for each area and a composite grid with 42 traits was created (Figure 2). Each student rated the topics against the traits. In addition, the students assigned relative weights (from 1 to 7) and offered preferred ratings for the criteria in general. There was an opportunity to discuss the meaning of ambiguous trait pairings prior to this exercise. Eight students participated in this phase.



Figure 2.

A composite rating grid for course content, including criteria developed by all seven students.

The individual ratings were then entered into Aquinas and grids were constructed for the course content. Ideally, all data would have been entered by the students, but the students had not received any instruction in the use of Aquinas. The data was analyzed individually and collectively.

First a set of preferences was determined by calculating a mean weight and a preference for each trait. This is illustrated in Figure 3. This table reveals much information about the students' views. For example, it is clear that students have a marked preference for technical issues over those of managing issues. They also prefer systems development issues over communication issues. Furthermore, they have a marked preference for practical issues over those perceived as theoretical.



Figure 3.

A table produced by Aquinas showing combined average weights (first column) and combined trait preferences from all seven students.

A consultation produced by Aquinas, using the preferences (see Figure 3), showed the degree of overall consensus and identified the correlation scores for all the students (Figure 4). Since the consultation values for each solution are fairly closely spaced, it shows the instructor that the topics are all relatively important to the students. It also echoes what was suggested before, a general preference for covering the early phases of development (for example, knowledge acquisition and knowledge representation) as opposed to covering later phases of development and management issues. A dissenting opinion is shown in Figure 5. The basic lesson in this table is that AI tools are more important to STUDENT-2 than to the rest of the class. The present way of having students make presentations on topics of personal interest allows individual strong interests to be explored. Interestingly, the issues of "Managing AI Software Projects" and "Technology Transfer" are the least important for both the consensus group and the dissenter.

Consensus Test Results for consultation
KE-CLASS-EVAL -- TEST-A0050

1: KNOWLEDGE ACQUISITION (.0.68)
2: KNOWLEDGE REPRESENTATION (0.67)
3: EXPLANATION (0.66)
4: SYSTEM DESIGN (0.66)
5: UNCERTAINTY (0.65)
6: NL AND OTHER ADVANCED USER INTERFACES (0.63)
7: AI TOOLS (0.61)
8: MANAGING AI SOFTWARE PROJECTS (0.54)
9: TECHNOLOGY TRANSFER (0.53)

Correlation scores for consultation consensus

STUDENT-6	0.80
STUDENT-5	0.88
STUDENT-4	0.70
STUDENT-3	0.38
STUDENT-2	0.33
STUDENT-1	0.85

Figure 4.

The results of a consultation in Aquinas using rating grids from all the students and the information in the table in Figure 3. The results show a consensus ranking of class elements by their relative importance. Correlation scores comparing each student to the consensus are then listed. STUDENT-2 has the lowest correlation score with the consensus opinion and, hence, has the most dissenting opinion. STUDENT-2 and STUDENT-3 have similar low correlation scores, and may have similar concerns that would be revealed through inspection of their dissenting opinions or difference grids (discussed later in this paper).

STUDENT-2 has the most dissenting opinion
(0.33 correlation score)

STUDENT-2		Consensus	
0.67	EXPLANATION	0.68	KNOWLEDGE ACQUISITION
0.62	AI TOOLS	0.67	KNOWLEDGE REPRESENTATION
0.62	UNCERTAINTY	0.66	EXPLANATION
0.60	NL AND OTHER ADV USER INTERFACES	0.66	SYSTEM DESIGN
0.60	KNOWLEDGE REPRESENTATION	0.65	UNCERTAINTY
0.60	KNOWLEDGE ACQUISITION	0.63	NL AND OTHER ADV USER INTERFACES
0.56	SYSTEM DESIGN	0.61	AI TOOLS
0.46	MANAGING AI SOFTWARE PROJECTS	0.54	MANAGING AI SOFTWARE PROJECTS
0.46	TECHNOLOGY TRANSFER	0.53	TECHNOLOGY TRANSFER

Figure 5.

The student with the lowest correlation score (see Figure 4) has the most dissenting opinion. Examining the side-by-side listing of the consensus and dissenting opinion, the instructor can get an idea of the range of opinion among the students.

Both trait and solution distance analyses were produced for all students (STUDENT-2 is shown in Figures 6 and 7). Cluster analysis shows how closely traits (or solutions) are related to each other (Boose and Bradshaw, 1987). In this process, the trait cluster analysis could be used in several ways. One use is to design lessons that address traits that are clustered together. Another use is to help produce a simplified questionnaire for use in the next class. The solution cluster analysis has several uses, mostly related to the structuring of information. This analysis suggests ways in which a single class could be structured around related topics, as well as ways in which a class might be broken into two or more classes.

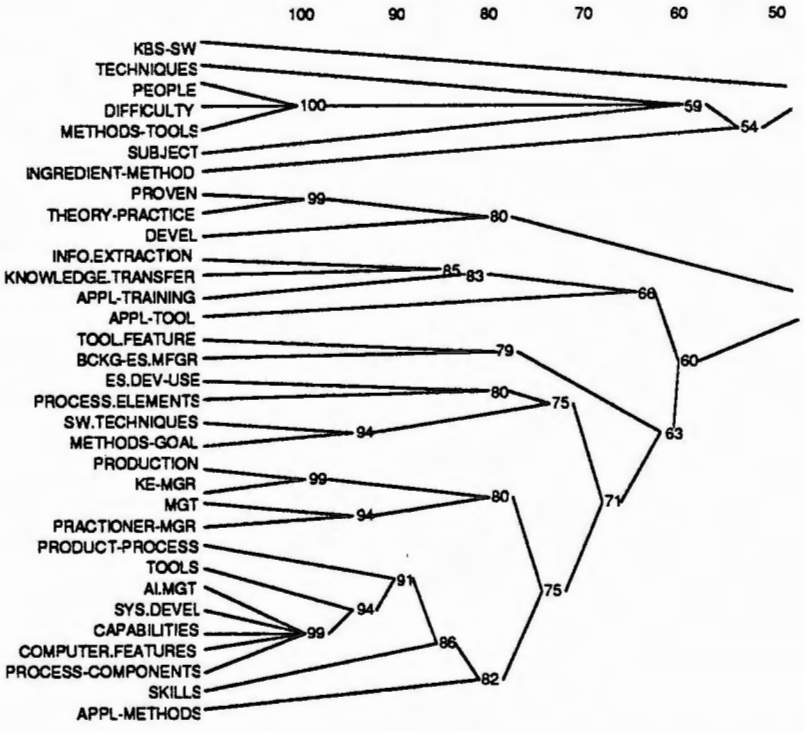


Figure 6.

A cluster analysis produced by Aquinas of evaluation criteria based on STUDENT-2's rating grid. This display showed relevant groups of criteria.

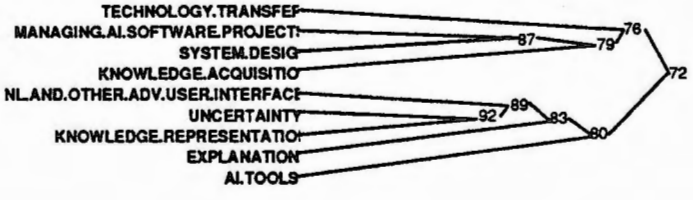


Figure 7.

A cluster analysis, produced by Aquinas, of course topics based on STUDENT-2's rating grid. This display suggested new ways to group course topics.

Next, a *difference grid* (based on MINUS, Shaw, 1980) was constructed for each possible pairing of students (Figure 8). Difference grids show the areas of greatest agreement and disagreement for a pair of rating grids from two students. Measurements are computed by taking the absolute difference between two ratings for corresponding element-trait pairs. Large disparities can reveal a difference in perception about the element or about the meaning of the trait itself. Column totals reflect differences about ratings or definitions of particular elements, and row totals reflect differences about ratings or definitions of particular traits.

ELEMENTS

1	2	3	4	5	6	7	8	9		
	.16	.66	.16	.50		.66	.33	.16	.29	AI
.16		1.00	.50		.50	1.00	.16		.37	AI.MGT
	.66	.50	.33			.50	1.00	.66	.40	APPL-METHODS
	.66	.50	.50	.50	1.00	.83	.50	.16	.51	APPL-TOOL
	.16		.50	.83			.50	.66	.29	APPL-TRAINING
.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	ATC
.50	.50	.50	.50	.50	.50	.50	.50	.50	.50	BCKG-ES.MFGR
.66	.33	.16	.33	.50	.33		.50	.50	.37	CAPABILITIES
.50			.83					1.00	.25	COMPUTER.FEATURES
	.16	.83	.16	.33	.16	.83		.66	.35	DEFINED
.16	.16	.66	.66	.16	.50	.33	.50	.16	.37	DEVEL
.83	.50	.66	.83	.50	.33	.33	.50	.83	.59	DIFFICULTY
.83	.50	.16		1.00	.50	1.00	.50	.16	.51	DIRECTION
.16	.16		.16	.50	.50		.50	.33	.25	ES.DEV-USE
.50	.50	.83	.33	.33	.33	.83		.83	.49	FOCUS
.16		.50	.66	.50		.16	.50	.16	.29	HELPS.USER
	1.00	.83	.83	.16	.33	.33		.33	.42	IDEAS
.83	.83		.83		.83	.66	.66	.16	.53	INFO.EXTRACTION
	.33	.50	.50			.33		.50	.24	INGREDIENT-METHOD
	.66	.66	.16	.33		.33	.50	.16	.31	KBS-SW
1.00	.33	.16	.16		.83	.66	.33	1.00	.50	KE-MGR
.50	.16		1.00	.50	1.00	.50		1.00	.51	KNOWLEDGE.TRANSFER
.50	.50	.16	.50	.50	.50	.16	.50	.50	.42	METHODS-GOAL
	.50	.50	.83	.33	.16		.50	.83	.40	METHODS-TOOLS
.66	.50	.50	.33		.33	.50	.50	.50	.42	MGT-PRODUCTION
.66	.50		.83	.33	.83		.16	.50	.42	PEOPLE
.83	.16								.11	PRACTICIONER-MGR
1.00	.33	.33	.83	.16	1.00	.33	.66	1.00	.62	PROCESS-COMP
.16	.83	.50	.16		.16			.33	.24	PROCESS.ELEMENTS
	.16	.83	.50			.83		.16	.27	PROGRAM
.16	.33		.33	.50	.16		.50	.33	.25	PROJECT-PROCESS
	.33	.16	.66	.66	.66	.16	.16	.83	.40	PROVEN
.66	.50	.33	.33	.16	.66	.83		.16	.40	SKILLS
.50	.33	.66	.83	.16	.16	.83	.33	.33	.46	SUBJECT
.50	.50	.16	.16	1.00		.66	.16	.16	.37	SW.TECHNIQUES
.66	.16		.50	1.00	.50		.50	.50	.42	SYS.DEVEL
1.00	.33	.83		.83	.33	.16	.83	1.00	.59	TECHNICAL
.50	.50	.16			.50	.33	.33		.25	TECHNICAL-POLITICAL
.50	.50	.83	.16	.83	.50	.66	.50		.50	TECHNIQUES
	.50		.50	.50	.16	.16	.16	.66	.29	THEORY-PRACTICE
.50		.16		.33		.50	1.00		.27	TOOL.FEATURE
	.33	.83	.16			.50	.83	.16	.31	TOOLS
.38	.38	.39	.43	.35	.35	.40	.37	.44	(0.39 avg per cell)	

1. AI TOOLS (.38)
2. EXPLANATION (.38)
3. KNOWLEDGE ACQUISITION (.39)
4. KNOWLEDGE REPRESENTATION (.43)
5. MANAGING AI SOFTWARE PROJECTS (.35)
6. NL AND OTHER ADV USER INTERFACES (.35)
7. SYSTEM DESIGN (.40)
8. TECHNOLOGY TRANSFER (.37)
9. UNCERTAINTY (.44)

Figure 8.

A difference grid showing individual differences and similarities between two students. In this case, the students agreed the most about *practitioner-manager*, *ingredient-method*, and *processing-elements* and disagreed most about *difficulty*, *process-components*, and *technical*. A blank space in the grid indicates zero difference, that is, total agreement; a 1.00 indicates greatest possible disagreement.

Difference and similarity scores were combined for all pairs of students, and the results were presented to the class. Three traits identified by Aquinas as exhibiting the most disagreement were used in a structured negotiation session. The session highlighted was conducted in which areas of strong agreement and disagreement. A blank rating was presented and the participants were asked to provide group ratings. Students were encouraged to query each other about trait definition and offer arguments for particular ratings. At this point the trait list was modified. Traits judged by the participants to be redundant, ambiguous, irrelevant, or unimportant were removed. The revised trait list was then again scored individually by the students. The purpose of the discussion was to get rid of duplication and ambiguity and isolate "true" disagreement, and to provide additional feedback to the instructor, who observed the negotiation session.

These ratings were entered into Aquinas and the analysis (consensus preference and difference rating) was performed a second time. Cluster analyses of the traits and solutions were constructed for each student.

Process Evaluation

In evaluating the success of the experiment, he must first consider the effectiveness of the evaluation. There are two considerations: (1) Was the information derived from the process useful to the task? (2) How useful was the process in producing the information? Together these two considerations form a rough measure of efficiency and effectiveness. If the derived information was useful and the process rapid and easy to accommodate, then the efficiency is high. Conversely, if the process is rapid and accommodating, but the knowledge unusable, then the efficiency is low. Efficiency can be further examined by viewing it from both the students' and instructors' perspectives. This determination will help us meet our objectives of assessing the effectiveness of using Aquinas for this task. Finally, we compare and contrast the process in terms of more traditional techniques.

Instructor's viewpoint

To evaluate the process, the instructor of the Knowledge Engineering class needed feedback on two key items. He needed to know generally (1) whether the content of the class was worthwhile - whether the topics themselves were interesting and useful; and (2) whether the format was appropriate to convey that content. If either the content or format was found to be deficient, the instructor would like to know the reasons and the suggested remedies. The instructor was also interested in recommendations for future classes. These could include changing the amount of time spent on a topic, changing the approach spent on a topic, adding or deleting topics, adding or deleting approaches, combining topics, combining approaches, or changing the order that topics are covered. Other possibilities include scrapping the class or adding another class to the curriculum.

Each analysis produced by Aquinas offered information of benefit to the instructor in some way. These were discussed as they arose in the text. This process showed that knowledge acquisition and analysis tools for individuals can be used to good advantage for groups as part of class evaluations.

Finding patterns in the the data offers many insights to the instructor. These patterns offer clues to what opinions are shared by what groups. For instance, Aquinas, through its clustering and classification capabilities shows that there is strong agreement about the need to cover knowledge acquisition. Conversely, the data shows that technology transfer is not perceived to be as important, in the context of the class, as other topics. This viewpoint appears to be shared by both the consensus and the dissenting opinions. Patterns in the data also reveal clues about how topics are related in the minds of the students, which offers useful insight as to how to structure content in future classes.

One important lesson is the need to present the data in as understandable a form as possible. It would be very useful to an instructor to be able to tell which suggestions and comments were shared by what percentage of the class. Was there a consensus on some items? On issues where strong dissension exists would it be possible to respond in some way that satisfied both or all groups? These measures will be added to a future version of Aquinas.

One obstacle to the process we used stems from unfamiliarity with the tool. It was not obvious to the instructor at first what could be done with the analysis provided by Aquinas. The task

confronting the instructor or decision-maker is two-fold: firstly, the numbers must be interpreted, secondly, they must be translated into action. This is not a significant problem. Experience with Aquinas sems to be the best remedy, although a good user interface and some user training are also important.

Student's Viewpoint

Students would like to convey their views to help the instructor correct what they see as course deficiencies. Because this is generally done as the last thing before walking out the door, and is seen as just another assignment (one that will not bias a grade), students would like to accomplish this as painlessly as possible. Evaluations may not be given their due respect, and be filled out hastily.

Traditionally, students do not see results from their evaluation efforts, because they generally do not repeat classes. Our process provides a some feedback to the students immediately via the structured negotiation session. The feedback is useful because students see their input actually considered. When several people from a single organization are taking a single class in succession (as is often the case in our classes), there may be a higher motivation to supply an evaluation which is likely to be acted on. In other collaborative scenarios feedback would play a larger role. Feedback is essential to participants who use the system, as it instills credibility in the system.

The strongest criticism of the process from the students' point of view was the time spent filling in grids with values. In the second phase, students supplied over 800 values for the composite grid ratings. As students were very conscientious about the values they supplied, this was quite time-consuming, taking up to two hours to complete. The amount of effort should correspond favorably to the benefits received. The instructor clearly benefits, but what about the students? Students commented to us that this was an interesting exercise, and are anxious to hear about the results. They also commented that there was a lot of work involved.

We would like to streamline the process while preserving its objectives. The size of grids could be reduced. This could be accomplished in a number of ways. One would be for the people conducting the experiment to eliminate seemingly redundant or ambiguous traits. This was not done for this process because we wanted to be unbiased with the data collection and analysis. Another possibility would be to present Aquinas analyses to students before the structured negotiation. Another likely possibility would be to only use the most important traits as identified by students (as shown in Boose, 1986a). Paring down the number of traits seems feasible. Reducing the number of ratings by reducing the solutions may not be feasible. There is an additional problem resulting from the number of values to be entered. Since there are so many values to enter, users may place less importance on individual values.

Researcher's Viewpoint

This section evaluates both the process and the manner in which Aquinas was used from the point of view of the researchers.

The model of problem-solving used in Aquinas is that of multiple knowledge sources (experts) that work together in a common problem solving context (case) by selecting the best alternatives for each of a sequential set of decisions (solutions). Alternatives at each step are selected by combining relevant information about preferences (relativistic reasoning), constraints (absolute reasoning) and information (probabilistic reasoning). In instances where the set of decisions are ordered with respect to solution refinement, this paradigm is similar to one suggested by Clancey, who showed that many problems are solved by abstracting data, heuristically mapping higher-level problem descriptions onto solution models, and then refining these models until specific solutions

are found (Clancey, 1986). Methods for propagating information and preferences are described in (Bradshaw and Boose, 1988).

If there is agreement between the consensus opinions and the expectation of the users, users gain more confidence that these are valid recommendations. If there are wide differences of opinion between experts, the user can employ other tools in Aquinas to compare the appropriate rating grids to see where the experts' opinions differed. For instance, differences in ratings in grids can be compared where there are common solutions and traits (Figure 8) (based on the MINUS tool in Planet, Shaw and Gaines, 1986).

A consideration that arises when using computers to support collaborative work is the presentation of group opinions. Participants may have differences in vocabulary or definitions, differences in semantics, differences in concepts behinds traits, or differences in information. They may agree on all aspects except the ratings, which constitutes genuine disagreement. They may disagree on models of process (and yet still agree on final conclusions). A consensus view is useful, but it might obscure one or more viable, special-case dissenting views. If the capability exists to present dissenting views, how much dissent must there be to have the "right" to a minority report. Perhaps more than one dissenting report might be needed.

People tend to have opinions with varying force and for varying reasons. "Weak" opinions are those held with little conviction; they may not be as important to consider as those held strongly. Some opinions are based on misinformation, and may not be as important as informed opinions. Structured discussion sessions tend to change weak opinions and expose misinformation.

Comparison with Traditional Evaluation

Traditionally, students in the Knowledge Engineering class were given a standard ATC evaluation form to evaluate a class. This consists of 7 yes/no, multiple choice, and short answer questions about course content, three questions about the instructor and one question about the facilities. The students filled out the standard evaluation in addition to the grid based approach presented here.

Does the traditional approach yield information about "qualitative" suggestions (for example, should class segments be longer? Should the order of the topics be changed?). Neither approach actually asks students what they think should be done with the course, although the Aquinas approach implicitly addresses that issue. The standard evaluation allows "additional comments" to be written. This provides a method for a student to explicitly comment on areas not addressed by the questionnaire or on those areas where strong opinions are held. This, however, is ad hoc and may or may not be used to advantage. In the future, we may use a questionnaire combining pre-existing criteria (developed in previous classes using Aquinas) with triadic comparison to produce new, unique criteria. In this way, evaluation forms could always be tailored for individual groups and classes, combining the best aspects of traditional questionnaires with those containing student-generated criteria.

Discussion

The Process as a Collaborative Process

The collaborative process that we have outlined has several phases. Analyzing the approach in terms of activities at each phase allows us to analyze, refine and, possibly, streamline the process. The phases are pictured in Figure 9.

1. Instructor develops grid solutions
2. Students add traits and ratings
3. Composite Grids are produced and rated by students
4. Grids analyzed with Aquinas
5. Structured negotiation
6. Students rate consensus grids
7. Instructor performs analyses on consensus grid
8. Results are relayed to students

Figure 9.
The collaborative evaluation process is a sequence of events.

The phases listed above will serve as an initial cut at our objective of defining a set of collaborative work processes. As we identify other types of collaborative tasks, we will expand the set of work processes which will help in defining new roles for Aquinas as a tool in an iterative, collaborative process.

The process is both explicitly and implicitly collaborative. It is explicitly collaborative during the structured negotiation session and implicitly collaborative during the Aquinas sessions where the exchange grids are analyzed.

Suchman and Trigg (1986) outline several issues that arise through collaborative relationships. These include the emergence of collaborative relationships, differences and synthesis, role specialization and interchangeability, technology and resources, forms of communication, and products. It is useful to consider these issues as they apply to the process used in this experiment. One of the primary aims of our approach is to highlight differences and to promote synthesis of various viewpoints. Role specialization and interchangeability could arise in our approach if different weighting schemes were used. For example, a specialist in one area, or members from a specific organization, could have relevant traits in their grid weighted more heavily. The technology and resources of this process have been discussed previously. Suchman and Trigg list three communication forms which are used in collaborative settings: substantive, annotative, and procedural. The major form in our process was substantive: most communication related directly to the matter at hand. The structured negotiation session was largely annotative, in which the meanings of traits were discussed and clarified. Procedural communication, communication about the process itself, was not integral to the process, although there was some discussion between the researchers and the students as to the effectiveness of the process. Finally, the product which resulted from the process, namely, individual, consensus, and dissenting grids, could form a part of a corporate memory.

The first phase of our process in which viewpoints are elicited anonymously resembles the Delphi technique in spirit. Our process, however, is more structured in its input and therefore more amenable to computer support. The second phase, commencing with the structured negotiation process, is quite different from the Delphi technique. Malone et al (Malone, 1986) refer to "semi-structured messages" to facilitate cooperation. Semi-structured messages are defined as "messages of identifiable types with each type containing a known set of fields, but with some of the fields containing unstructured text or other information." Interestingly, the process that we tested uses

semi-structured messages which are conversely related to the one that Malone et al describe, that is the fields (the solutions and traits) can vary but the contents (the ratings) are strongly typed.

Our process was used in a situation of low conflict, in which there is no contention for resources or other zero-sum situations. This is shown based on values in the difference grids. Stephanou and Lu (1984), working from an information theoretic perspective, have shown that information from multiple, as opposed to single, individuals greatly enhances the overall result, even in situations of only moderate consensus.

Future Directions

There are several other possible uses for this process within the class evaluation context. One use would be to have students use Aquinas directly. The analysis could then be used to help generate new class segments. Another use of the combining of viewpoints is suggested by the need to compare evaluations from different classes. Instructors would like to know whether changes made as a result of student input changed the values in the grids. It might also be possible to use this process before the class started to concentrate on areas most needed by students. Another use of this technique would be a revamping of the course evaluation forms presently in use. It would also be useful to note how evaluations changed over time.

We are considering other opportunities both in extending Aquinas and in understanding and supporting collaborative processes. Extensions to Aquinas include report writing and other summarizing capabilities, and more grid-combining activities. Another capability under consideration is an on-line, parallel, multi-user, dynamic grid entry and analysis capability similar to Chang's Participant System (Chang, 1986). We are also looking towards using Aquinas as one tool in a collaborative process workbench. The workbench would integrate many current ATC research projects and other commercial products into distributed graphic "coordination" systems (see, for example Holt, 1983 and Winograd, 1986).

Acknowledgements

Thanks to Kit Bakke, Miroslav Benda, Keith Butler, Stan Covington, Cathy Kitto, Art Nagai, Mary Park, and Dave Shema. We would also like to thank the students of the ATC Knowledge Engineering and Knowledge Acquisition classes who participated in this experiment enthusiastically while being inundated with course assignments. This work was performed at the Advanced Technology Center of Boeing Computer Services in Bellevue, Washington.

References

J. Boose, "Personal Construct Theory and the Transfer of Human Expertise", *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, 1984

J. Boose, "Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain", *Future Computing Systems*, Vol. 1 Number 2, 1986a

J. Boose, *Expertise Transfer for Expert System Design*, Elsevier, 1986b

J. Boose and J. Bradshaw, "Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, Special Issue of *International Journal of Man- Machine Studies* on the AAAI Knowledge Acquisition for Knowledge- Based Systems Workshop, 1987

J. Boose 1988, "Uses of Repertory Grid-Centered Knowledge Acquisition Tools for Knowledge-Based Systems", *International Journal of Man-Machine Studies*, in press

J. Boose, D. Shema, and J. Bradshaw, "Recent Progress in Aquinas: A Knowledge Acquisition Workbench", Proceedings of the Third AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada, November 1988.

J. Bradshaw and J. Boose, "Decision Analytic Techniques for Knowledge Acquisition: Combining Information and Preferences using Aquinas", *International Journal of Man-Machine Studies*, in press

J. Bradshaw, J. Boose, S. Covington, and P. Russo, "How to Do with Grids What People Say You Can't", Proceedings of the Third AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems, Banff, Canada, November 1988.

E. Chang, "Participant Systems", Future Computing Systems, Vol. 1, Number 3, 1986

W. Clancey, "Heuristic Classification" in *Knowledge-Based Problem Solving*, J. Kowalik (ed), Prentice-Hall, 1986

O. Helmer, "The Use of the Delphi Technique", Rand Corporation, 1966

A. Holt, H. Ramsey, and J. Grimes, "Coordination System Technology as the Basis for a Programming Environment", *Electrical Communication*, Volume 57, Number 4, 1983

G. Kelly, *The Psychology of Personal Constructs*, Norton, 1955

C. Kitto and J. Boose, "Heuristics for Expertise Transfer: An Implementation of a Dialog Manager for Knowledge Acquisition", Special Issue of *International Journal of Man-Machine Studies* on the AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987

T. Malone, K. Grant, K. Lai, Ramana Rao, and D. Rosenblitt, "Semi-Structured Messages are Surprisingly Useful for Computer- Supported Coordination", *Proceedings of the Conference on Computer-Supported Cooperative Work*, Austin, TX, 1986

M. Shaw, *On Becoming a Personal Scientist*, Academic Press, 1980

M. Shaw and B. Gaines, "Techniques for Knowledge Acquisition and Transfer", *International Journal of Man-Machine Studies*, Vol. 27, No. 3, 1986

L. Suchman and R. Trigg, "A Framework for Studying Research Collaboration", *Proceedings of the Conference on Computer- Supported Cooperative Work*, Austin, TX, 1986

T. Winograd, "A Language/Action Perspective on the Design of Cooperative Work", *Proceedings of the Conference on Computer- Supported Cooperative Work*, Austin, TX, 1986

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

A Methodology for Recognizing Consensus, Correspondence, Conflict and Contrast in a Knowledge Acquisition System

Mildred L. G. Shaw and Brian R Gaines
Knowledge Science Institute
Department of Computer Science
University of Calgary
Calgary, Alberta,
Canada T2N 1N4

ABSTRACT

One problem of eliciting knowledge from several experts is that experts may share only parts of their terminologies and conceptual systems. Experts may use the same term for different concepts, use different terms for the same concept, use the same term for the same concept, or use different terms and have different concepts. Moreover, clients who use an expert system have even less likelihood of sharing terms and concepts with the experts who produced it. This paper outlines a methodology for eliciting and recognizing such individual differences. KSS0 is a knowledge support system providing an integrated set of tools for knowledge acquisition and the rapid prototyping of knowledge-based systems. *Socio* is a tool in KSS0 that compares the conceptual systems of multiple experts in the same domain. It can be used to focus discussion between experts on those differences between them which require resolution, enabling them to classify them in terms of differing terminologies, levels of abstraction, disagreements, and so on. Socio promotes the full exploration of the conceptual framework of a domain of expertise by encouraging experts to operate in a "brain-storming" mode as a group, using differing viewpoints to develop a rich framework. It avoids social pressures forcing an invalid consensus by providing objective analysis of separately elicited conceptual systems.

INTRODUCTION

The elicitation of knowledge from one expert presents enough problems that elicitation from many might seem an unnecessary complication. However, a major reason for practical interest in expert systems is that in many domains, such as quality control, expertise is essentially distributed over many experts, and the purpose of the system is to bring it together (Hayes-Roth 1984). More fundamentally, the use of multiple experts in 'brain-storming' or synectics sessions has been found valuable in problem-solving, and is an attractive technique to prevent individuals experts 'blocking' and failing fully to explore and express their conceptual domains.

It is important to note in dealing with multiple experts that consensual agreement upon domain concepts is only one of many significant possibilities. Experts may legitimately have different terminologies for the same domain concept. They may describe it at different levels of abstraction. One expert may describe a concept in operational terms and another in descriptive terms. They may also legitimately use the same terminology for different domain concepts. They may be using the same term distinguished by different contexts. One expert may have a very different conceptual framework or strategy from another. These differences may be carried through to an expert system design that allows users to obtain advice based on different and mixed sources of expertise, and to ask for dissenting opinions (Boose & Bradshaw 1987).

Thus, in a knowledge acquisition system, it is important not to attempt to force a false consensus on a group of experts on the assumption that there is some 'correct' terminology and conceptual framework. However, it is also important to bring to light differences among the experts and make these clearly available for discussion. Some of them may reflect errors in elicitation, others differences in terminology, others differences in conceptual frameworks. In any event, the discussion of these differences is in itself a significant stage in the knowledge elicitation process.

The next section establishes a theoretical framework for knowledge elicitation from mutiple experts. The following sections describe a practical methodology based on this framework, a computer tool implementing this methodology, and results of using this tool to apply the methodology to practical knowledge elicitation.

CONCEPTUAL SYSTEMS IN A DOMAIN COMMUNITY

Figure 1 shows a domain in which an expert system could be constructed from the knowledge of existing experts. The two "experts" shown, Expert 1 and Expert 2, are best thought of as *roles* in the individuals involved. These expert roles are distinguished by involving conceptual systems which, when applied to problem solving in the domain, lead to recognition of the individuals playing the roles as beings "experts" in the domain. The individuals will also have other roles involving other conceptual systems appropriate to other domains in which they may, or may not, be experts.

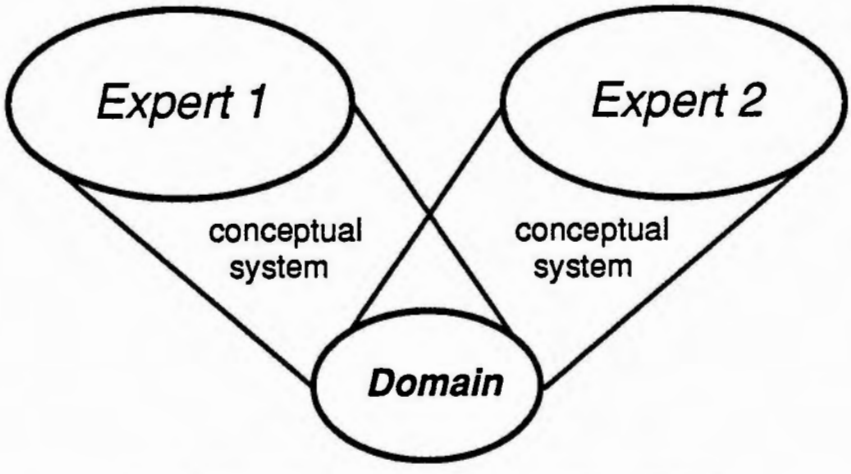


Fig.1 Experts acting in the same domain

The figure shows an overlap in the two conceptual systems indicating corresponding concepts. The experts will each have terminologies for expressing the concepts involved. The possible overlaps between concepts and terminologies leads to four relations which can occur between parts of the conceptual systems as shown in Figure 2:

- *Consensus* occurs when there is agreement in both the concepts and terminology used;
- *Correspondence* occurs when different terms are used for the same concepts;
- *Conflict* occurs when the the same terms are used for different concepts;
- *Contrast* occurs when different terms are used for different concepts.

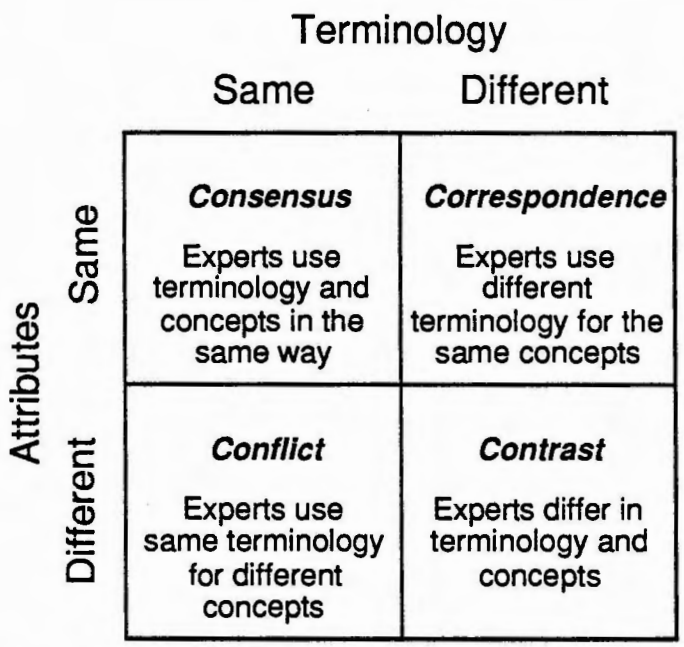


Fig.2 Consensus, correspondence, conflict and contrast among experts

The recognition of consensual concepts is important because it establishes a basis for communication using shared concepts and terminologies.

The recognition of corresponding concepts is important because it establishes a basis for mutual understanding of differing terms through the availability of common concepts.

The recognition of conflicting concepts is important because it establishes a basis for avoiding confusion over the labeling of differing concepts with same term.

The recognition of contrasting concepts is important because it establishes that there are aspects of the differing expertise about which communication and understanding may be very difficult, even though this should not lead to confusion. Such contrasts are more common than is generally realized. For example, it is possible to derive the same theorem in mathematics either by using an algebraic perspective, or a geometric one. There is nothing in common in these two approaches except the final result. It may still be possible to discuss the same domain using consensual and corresponding concepts that were not fundamental to the problem solving activities.

These considerations also apply to clients of the experts whose concepts and terminologies may be evaluated along these four dimensions in relation to those of the expert. The recognition of possible conflicts between the experts' and clients' use of terminology, and the provision of a variety of corresponding concepts, are major factors in the usability of an expert system.

In Figure 3, Client 1 is shown as having overlap only with Expert 1. He or she has no overlap whatsoever with any other individual in the figure, expert or user. Client 2 has overlap both with Client 3 and with the two experts. He or she is able to understand the concepts of Expert 2 and to a lesser extent those of Expert 1. Client 3 has no overlap with either expert, but has sufficient overlap with Client 2 that Client 2 can explain to him the advice given. In the development of an expert system, Client 1 may choose not to take advantage of Expert 2's expertise in the system. It would be particularly important to develop additional concepts and terminology to enable Client 3 to have adequate communication with, and understanding of, the system.

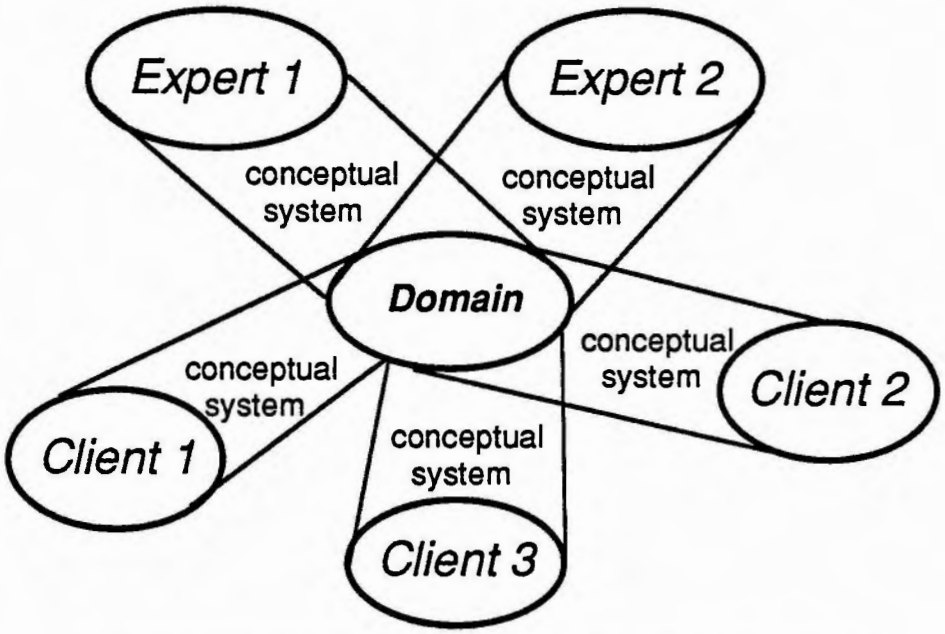


Fig.3 Experts and clients operating in the same domain

These are some of the possible relations which may occur between the conceptual systems of users and experts. The model is one of *roles* within an individual using a particular *conceptual system* and expressing it in a particular way to enter into an externally perceived and valued *domain community*. This model of coherent intellectual processes, each with its own individuality, each using its own conceptual system and its own terminology to enter into the domain community provides a cogent picture of the rationale behind, and the conflicts involved in, the development of communication and understanding in the community and in expert systems playing a role in that community.

DERIVING CONCEPTUAL SYSTEMS

Personal Construct Psychology (Kelly 1955, Shaw 1980) has been widely used as a basis for developing methods of knowledge acquisition from experts in a given domain (Shaw & Gaines 1983, Boose 1984, Diederich, Ruhmann & May 1987). Personal Construct Psychology is a psychology of the individual and many of its applications emphasize the idiosynchratic nature of conceptual systems (Mancuso & Shaw 1988). However, it is also a psychology of the individual interacting with the world and embedded in society (Shaw 1985). It encompasses shared concepts and conceptual systems, and their generation through experience. In particular it encompasses those systems that are so widely shared and so significant that they are construed as *knowledge*, and treated as having an existence virtually independent of their carriers.

Entity-attribute, or repertory grid, methodologies based on Personal Construct Psychology allow a significant part of the conceptual systems of experts to be elicited through interactive interviewing techniques such as those of KSS0 (Gaines 1987) and Aquinas (Boose and Bradshaw 1987). The resultant data structure is one in which the terms for entities and attributes in a domain have been specified by the expert, together with the values of those entities along the dimensions of the attributes. The entities are usually concrete items in the domain whose nature, definition and names can be agreed by experts and clients. The attributes reflect individual conceptual systems and may be used, and labeled, idiosynchratically.

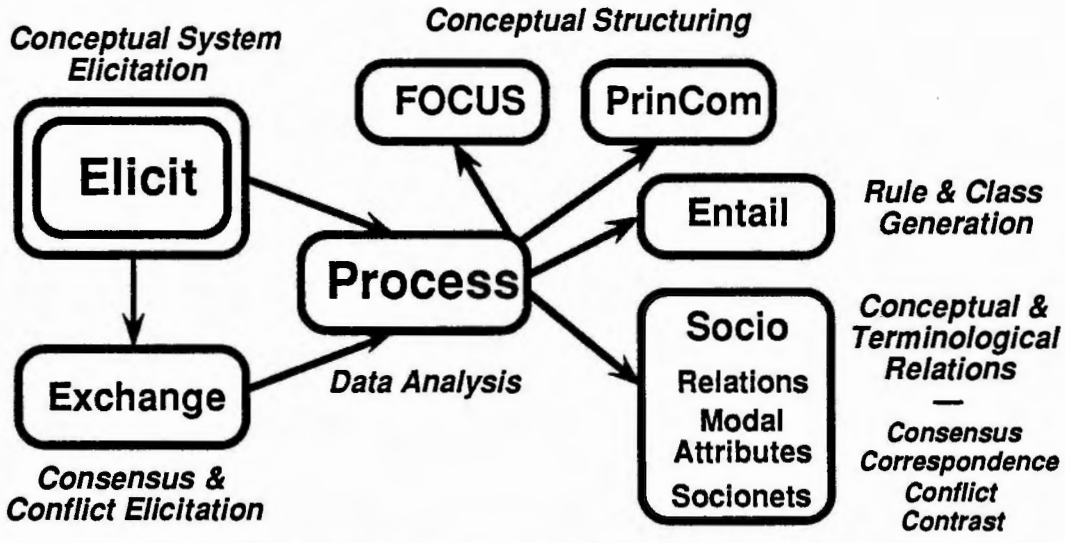


Fig.4 Some tools in Knowledge Support System Zero

Figure 4 shows the main tools in KSS0 relevant to the issues discussed in this paper:

- *Elicit* accepts specifications of entities within a domain and provides an interactive graphical elicitation environment within which the experts can distinguish entities to derive their attributes. The resultant conceptual system is continuously analyzed to provide feedback prompting the expert to enter further entities and attributes.
- *Exchange* allows the terms in the conceptual system derived from one expert to be used by another in order to determine whether the two experts have consensus or conflict in their use of terminology and concepts.
- *Process* gives access to various cluster analytic methods for the analysis and display of the conceptual systems elicited: FOCUS shows the system as a hierarchical structure; PrinCom as a spatial map; and Entail as a logical structure of classes and rules.
- *Socio* compares elicited and exchanged grids in a variety of ways to determine consensus, correspondence, conflict and contrast, and its methodology is described in detail in the next section.

Figures 5 and 6 give examples of the interactive graphical elicitation of attributes using KSS0 in a study of the consistency of expertise across experts, and across time, in a small group of geographers specializing in mapping techniques and their application to geological exploration (Shaw & Woodward 1987). Figure 5 shows some of the mapping techniques used as entities through a rating screen from the program, *Elicit*, in which a geographer is rating the entity, *trend surface analysis*, on the attribute, *doesn't incorporate geologic model—incorporates geologic model*. Figure 6 shows some of the characteristics derived as attributes through a match screen also from *Elicit* in which a geographer is being asked to distinguish *punctual kriging* from *universal kriging*. Figure 7 shows an elicited entity-attribute grid.

Exchange methodologies were developed for the measurement of understanding and agreement between either two individuals, two roles or on two occasions (Shaw 1980). To do this two people, possibly experts with differing points of view, each elicit a grid in an area of common knowledge or experience. Each may choose his own entities independently of the other, and elicit and rate his or her attributes quite separately. Each then can *Exchange* his or her grid, that is use the other's entities and attributes but fill in his or her own the rating values. For example, in terms of Figure 5, the exchanging expert would see the terms *doesn't incorporate geologic model—incorporates geologic model*, but the entities would all be to the left and have to be dragged to the scale with no knowledge of where the other expert had previously placed them.

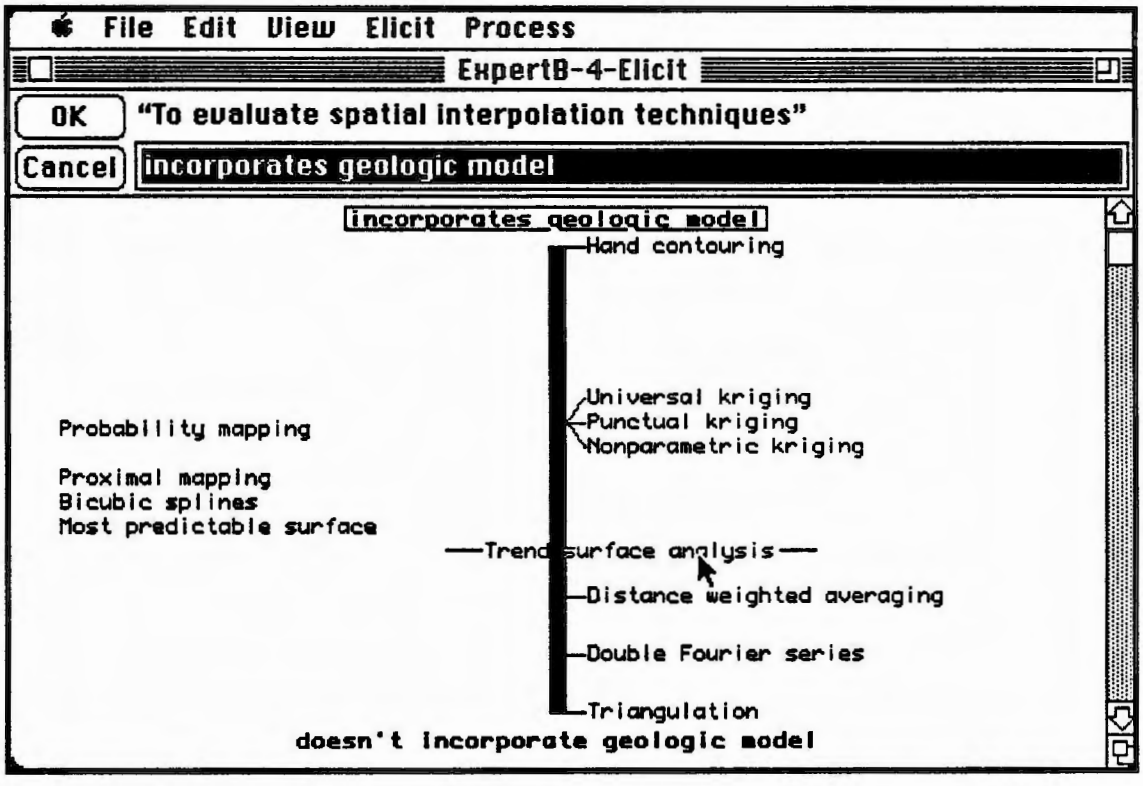


Fig. 5 KSS0 attribute elicitation screen showing entities being rated

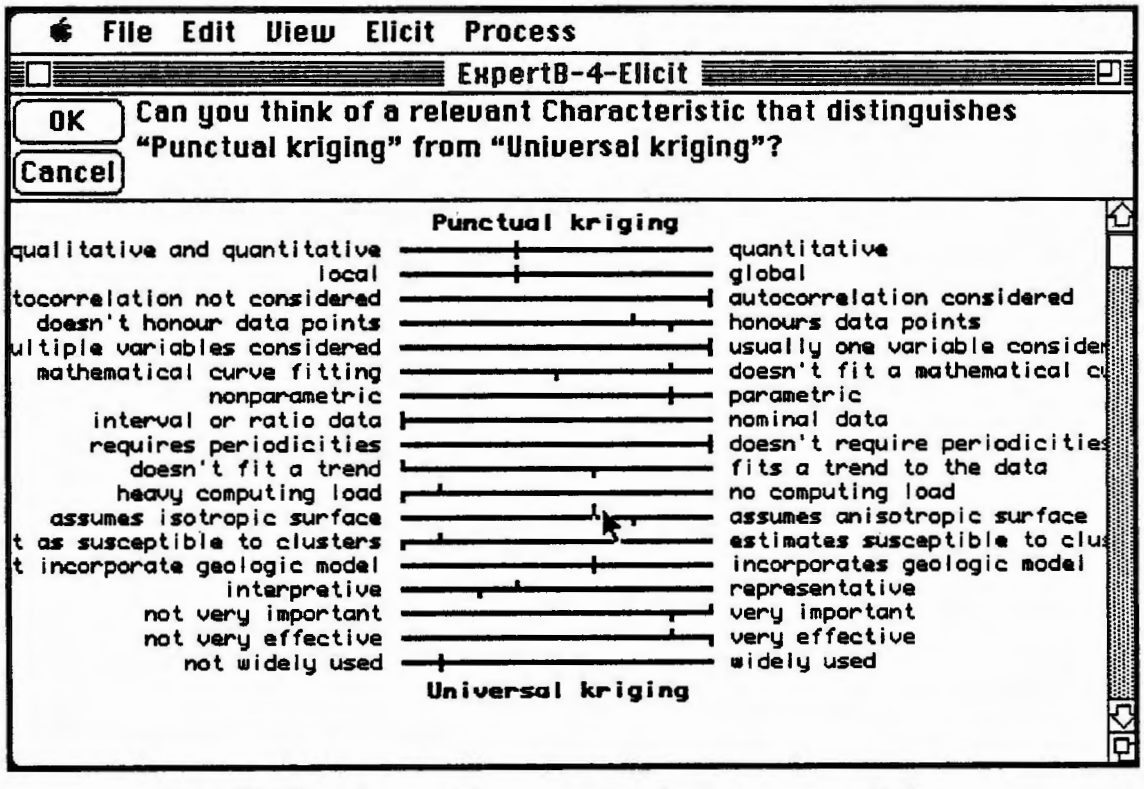


Fig. 6 KSS0 entity match screen showing ratings on attributes

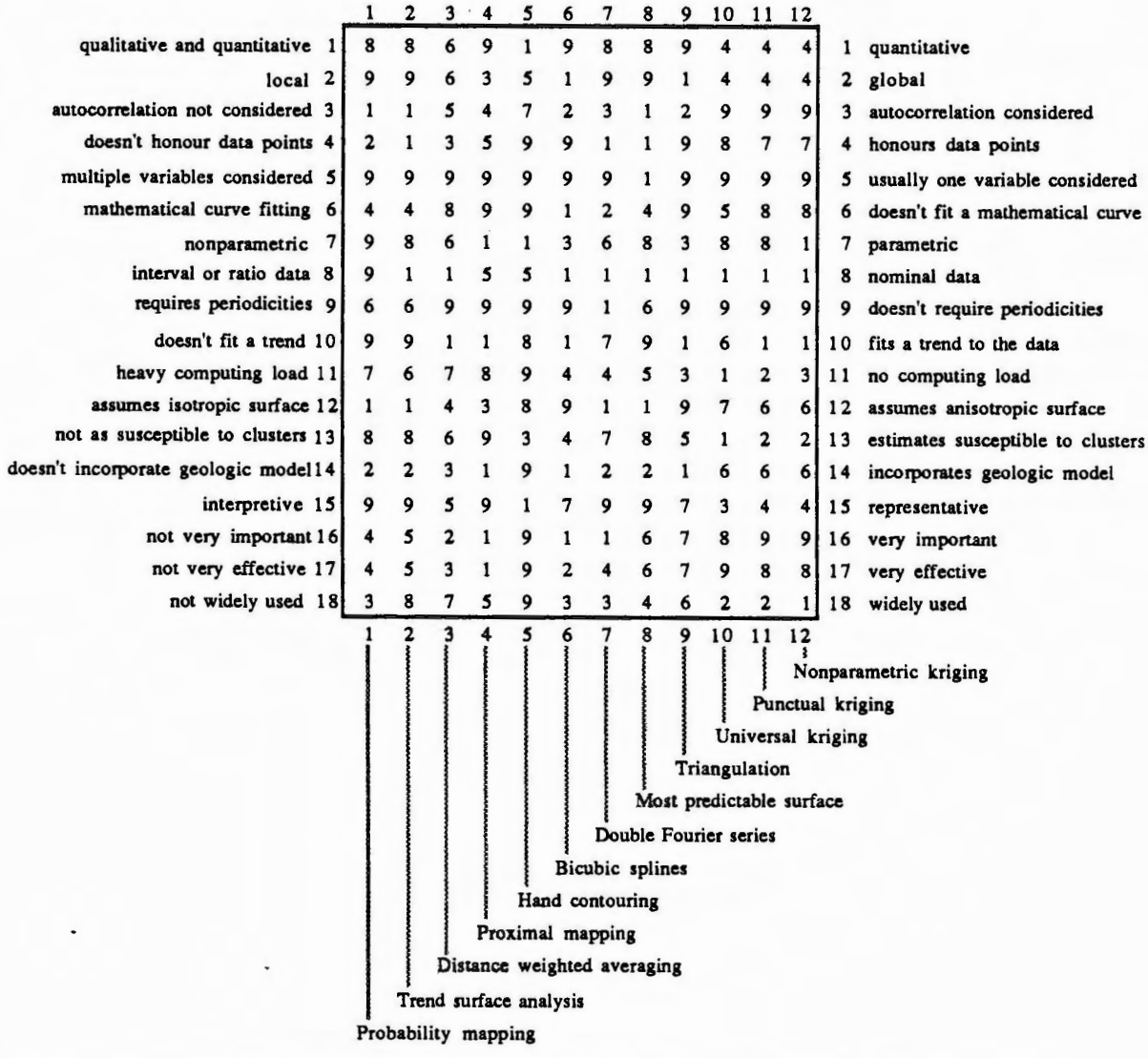


Fig.7 Conceptual system in entity-attribute grid

DERIVING RELATIONS BETWEEN CONCEPTUAL SYSTEMS

The *Socio* analysis in KSS0 allows members of a community to explore their agreement and understanding with other members, and to make overt the knowledge network involved (Shaw 1980, 1981, 1988). It is an extension of techniques such as SOCIOGRIDS (Shaw 1980) for deriving socionets and mode constructs from groups of individuals construing the same class of entities. The objective of Socio is to take different conceptual systems in the same domain and compare them for their structure, showing the similarities and differences. It may be regarded as the implementation of a simple form of analogical reasoning. Figure 8 shows the basis of operation of Socio—consider one set of data as being the base class defined by its entities, their attributes and values, and consider variant classes:

- *Entity-Attribute Compare*: The conceptual system at the upper left has the same entities and attributes but possibly differing values. Socio analyzes the matches between the entities and the attributes in this and the base class according to the values, and shows those entities and

attributes that are similar and those which are different. A typical application is to see whether experts agree on the definitions of attributes by asking them to separately fill in the values for a grid exchanged between them.

- *Entity Compare*: The conceptual system at the upper right has different entities but the same attributes. Socio analyzes the matches between the entities in this and the base class, and for each entity in the base class shows the closest matching entity in the other class. A typical application is to see whether experts are using different terminologies for the same entities by asking them separately to define entities and fill in the values for a domain defined through a class with agreed attributes.
- *Attribute Compare*: The conceptual system at the lower left has different attributes but the same entities. Socio analyzes the matches between the attributes in this and the base class, and for each attribute in the base class shows the closest matching attribute in the other class. A typical application is to see whether experts are using different terminologies for the same attributes by asking them to separately define attributes and fill in the values for a sub-domain defined through a class with agreed entities.

If, as in the lower right, neither entities nor attributes are common then no comparison is possible.

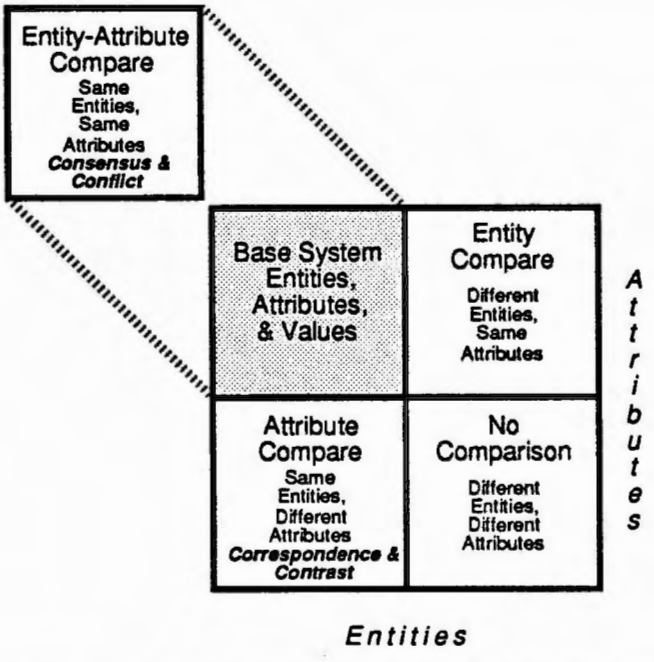


Fig.8 Possible comparisons between base and related systems

When a number of classes representing the same sub-domain are available, Socio also provides two other forms of analysis:

- *Mode Entities and Attributes*: Socio attempts to derive "modal" entities or attributes that reflect a consensus among the experts. It does this by extracting those which occur as highly matched entities or attributes across the majority of conceptual systems. A typical application is to reach consensus on critical concepts that are associated with a rich vocabulary at differing levels of abstraction.
- *Socionets*: Socio derives a socionet showing the degree to which each expert is able to make the same distinctions as another expert, even if they use different terminology. A typical application is to validate the knowledge acquisition process by determining whether the structure derived conforms with known relations between the experts.

CONSENSUS, CORRESPONDENCE, CONFLICT AND CONTRAST

Using the concepts developed above it is possible to develop a complete methodology for eliciting and analyzing consensus, correspondence, conflict and contrast in a group of experts, and implement this as an automatic process using the tools in KSS0. The methodology has three phases shown in Figures 9 through 11.

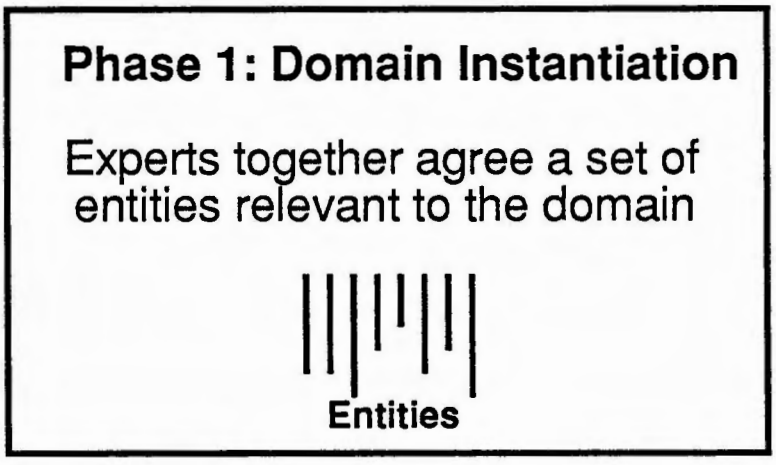


Fig.9 Co^4 Methodology Phase 1

Phase 1: Domain Discussion and Instantiation

In phase 1 a group of experts comes to an agreement over a set of entities which instantiate the relevant domain. This is the initial phase of any entity-attribute methodology, whether used with individuals or groups. However, with individuals the elicitation techniques may be used to elicit more entities as the exploration of the conceptual domain proceeds. When comparing multiple experts it is important that a set of entities is established at the start of the comparative study, and that the experts mutually agree on the definitions of these.

A convenient way to generate this set of entities is have each expert individually use Elicit to enter his or her conceptual system for a domain, and then extract the elicited entities from all the grids for discussion and consolidation by the group. This has the advantage that the experts gain some experience in the use of the KSS0 tools and can take advantage of the full elicitation facilities.

Phase 2: Conceptualization and Feedback

In phase 2 each expert individually elicits attributes and values for the agreed entities. The resultant conceptual systems will have the same entities but different attributes and can be analyzed by the *Attribute Compare* component of Socio as shown in Figure 8. This takes each attribute in one grid and determines the best matching attribute in the other grid, if there is one. The result is a mapping from the attributes in one expert's grid to those in another's as shown in Figure 10.

In evaluating this mapping we are not particularly interested in the terminology used but rather whether one expert has an attribute that can be used to make the same distinctions between the entities as does the other expert, regardless of whether these distinctions are called by the same terms. If such a *correspondence* occurs then the experts have a basis for mutual understanding of the underlying concept.

If an attribute in one system has no matching attribute in the other then it stands in *contrast* to all the other expert's attributes, and it may be very difficult for the other expert to understand the use of this term.

Note that the arrows in Figure 10 need not be symmetric. Attribute A in G1 may be best matched by attribute B in G2, but attribute B in G2 may be best matched by attribute C in G1. The only constraint is that if an attribute has an incoming arrow then it will have an outgoing arrow.

Thus, the second phase provides the basis for analysis of correspondence and contrast as already discussed.

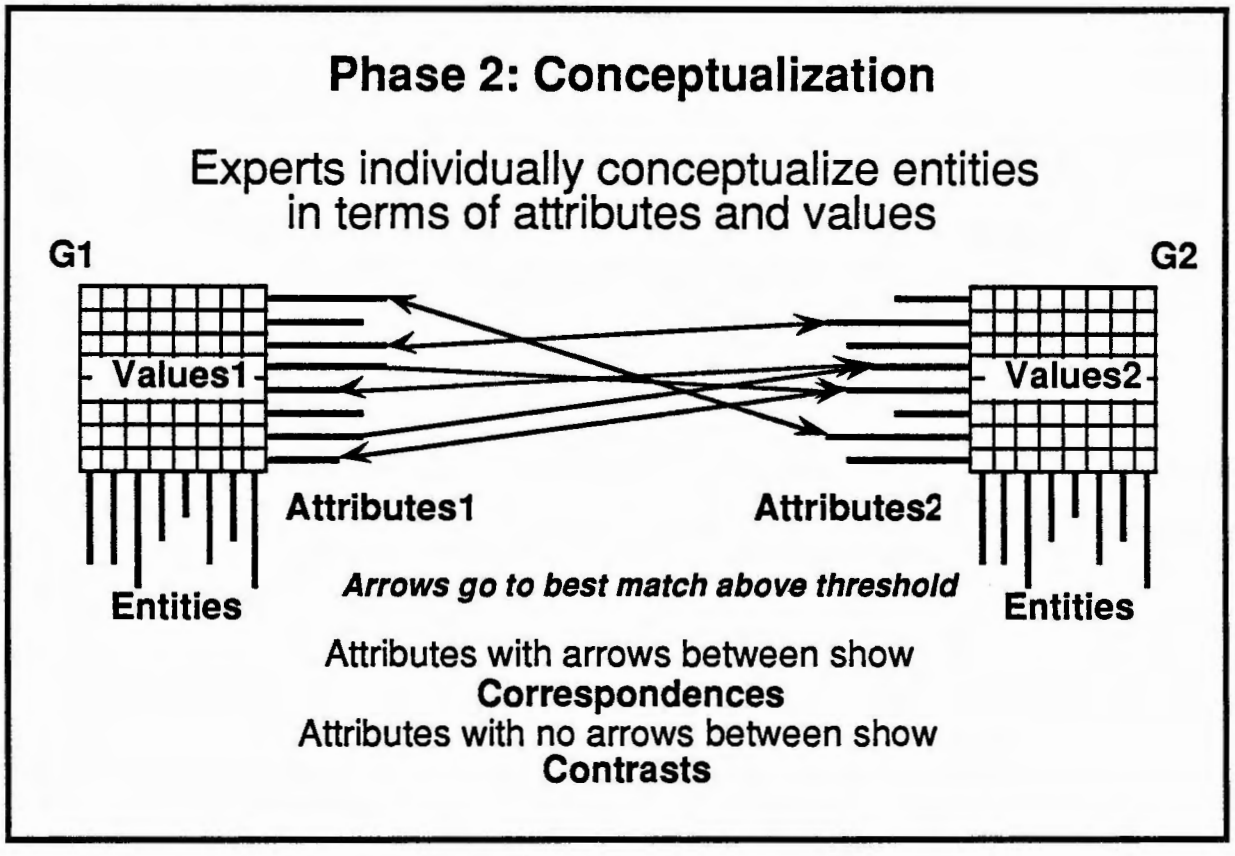


Fig.10 Co^4 Methodology Phase 2

Phase 3: Exchange and Compare

In phase 3 each expert individually exchanges elicited conceptual systems with every other expert, and fills in the values for the agreed entities on the attributes used by the other experts. The resultant conceptual systems will have the same entities and attributes and can be analyzed by the *Entity-Attribute Compare* component of Socio as shown in Figure 8. This takes each attribute in one grid and determines whether it matches the corresponding attribute in the other grid.

The result is a map showing *consensus* when attributes with the same labels are used in the same way and *conflict* when they are not as shown in Figure 11. Thus, the third phase provides the basis for analysis of consensus and conflict as already discussed.

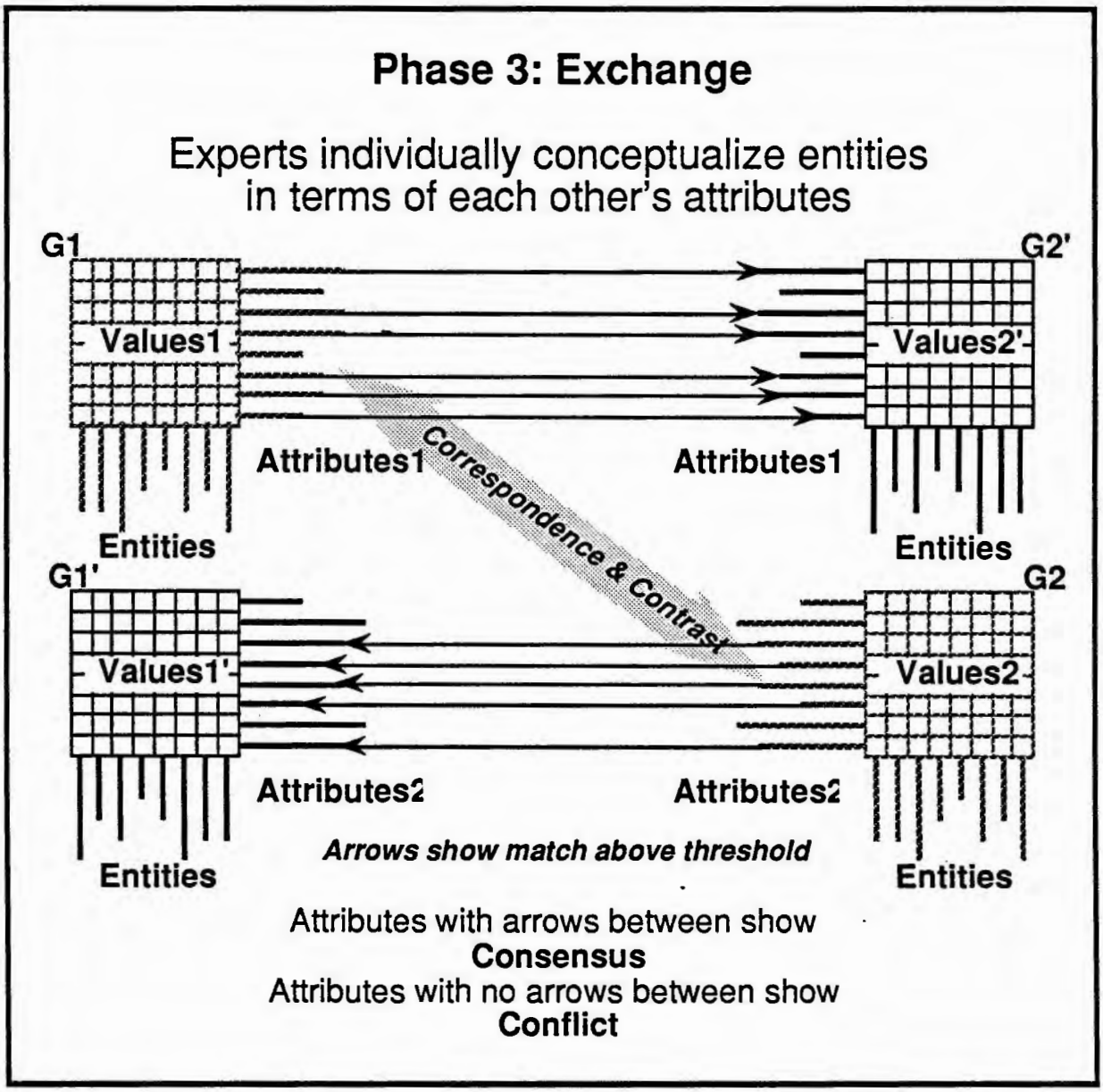


Fig.11 Co^4 Methodology Phase 3

Figure 11 also shows the correspondence and contrast relations analyzed in phase 2 as a relations between two of the grids used in phase 3. Thus, for two experts, four grids obtained by one elicitation and one exchange each, are sufficient to classify the relations between attributes in terms of consensus, correspondence, conflict and contrast. The methodology scales up linearly for each expert, so that n experts will be involved in n elicitations, one base elicitation and n-1 exchanges.

These three phases result in the experts' conceptual systems having become overt and inter-related. They lead naturally to later phases in which classes, objects and rules can be developed incorporating consensual, corresponding, and some of the contrasting attributes as kernel knowledge and, possibly, the conflicting and remaining contrasting attributes as 'other opinions.'

EXAMPLES OF THE METHODOLOGY IN ACTION

This section gives an example of the methodology in action using data from the study of geographers specializing in mapping techniques previously cited (Shaw & Woodward 1987).

Figure 12 shows an entity-attribute comparison from the geographic study in which expert B has filled in the values for a class defined by entities and attributes elicited from expert A. The print out shows the matches sorted with best first. The cumulative percentage is given of the number of attributes with matches greater than the value shown. In the list of attributes at the top, it can be seen that there is consensus on *interval data—nominal data*, but conflict on *requires no model—requires a model* and *linear interpolation—nonlinear interpolation*. There is clear consensus on the top four attributes, clear conflict on the lower five, and uncertainty about the remaining three.

In the list of entities at the bottom, it can be seen that there is close agreement on *vector trend analysis*, but high disagreement on *probability mapping*. This output can be used to focus a discussion between the experts on why they differ in their views of *probability mapping* and the classification of mapping techniques in terms of *linear* or *nonlinear interpolation*. For example, the first attribute, *requires no model—requires a model* shows high disagreement, and looking further into the elicited data it can be seen that expert A thinks that *probability mapping requires a model*, whereas expert B thinks that *probability mapping requires NO model*. On inquiring into this, the explanation given was couched in terms of what one actually means by the term "model", indicating the conflicting use of terminology.

```
G1:G2 33.3% over 80.0 (ExpertB attribute-consistency-with ExpertA)
  1:   8.3% ≥ 90.9   A2: Interval data - Nominal data
  2:          81.8   A4: Global - Local
  3:          81.8   A5: Intuitive - Mathematical
  4:  33.3% ≥ 81.8   A6: Requires spatial search - Does not require spatial search
  5:  41.7% ≥ 75.0  A10: Difficult to understand - Easily understood
  6:          72.7   A3: Non-polynomial - Polynomial
  7:  58.3% ≥ 72.7   A7: Discontinuous - Continuous
  8:  66.7% ≥ 59.1  A12: Does not consider non-spatial attributes - Considers non-spatial
  9:  75.0% ≥ 56.8  A11: Few points - Many points
 10:  83.3% ≥ 50.0   A8: Does not honour data - Honours data
 11:          47.7   A1: Requires no model - Requires model
 12: 100.0% ≥ 47.7   A9: Linear interpolation - Non-linear interpolation

G1:G2 8.3% over 80.0 (ExpertB entity-consistency-with ExpertA)
  1:   9.1% ≥ 83.3  E11: Vector trend surface analysis
  2:  18.2% ≥ 77.1   E2: Trend surface analysis
  3:          72.9   E4: Distance weighted averaging
  4:  36.4% ≥ 72.9   E7: Bicubic splines
  5:          70.8   E3: Kriging
  6:          70.8   E5: Proximal mapping
  7:          70.8   E8: Double Fourier series
  8:  72.7% ≥ 70.8   E9: Most predictable surface
  9:  81.8% ≥ 60.4  E10: Negative exponential surface
 10:  90.9% ≥ 56.2   E6: Hand contouring
 11: 100.0% ≥ 43.8   E1: Probability mapping
```

Fig.12 Entity-attribute comparison of expert B with expert A

Figure 12 can be redrawn as a *difference grid* where rating values (in this case 1 to 5) for expert B's ratings of expert A's entities on his attributes are subtracted from expert A's similar rating values respectively. Figure 13 shows this with the entities and attributes about which they agree the most in the top right corner, shown by no difference or a difference of only 1; and those with most disagreement towards the bottom left, shown by the maximum difference of 4 or a large difference of 3. Hence from this difference grid, the consensus and conflicts can easily be identified and discussed by the experts.

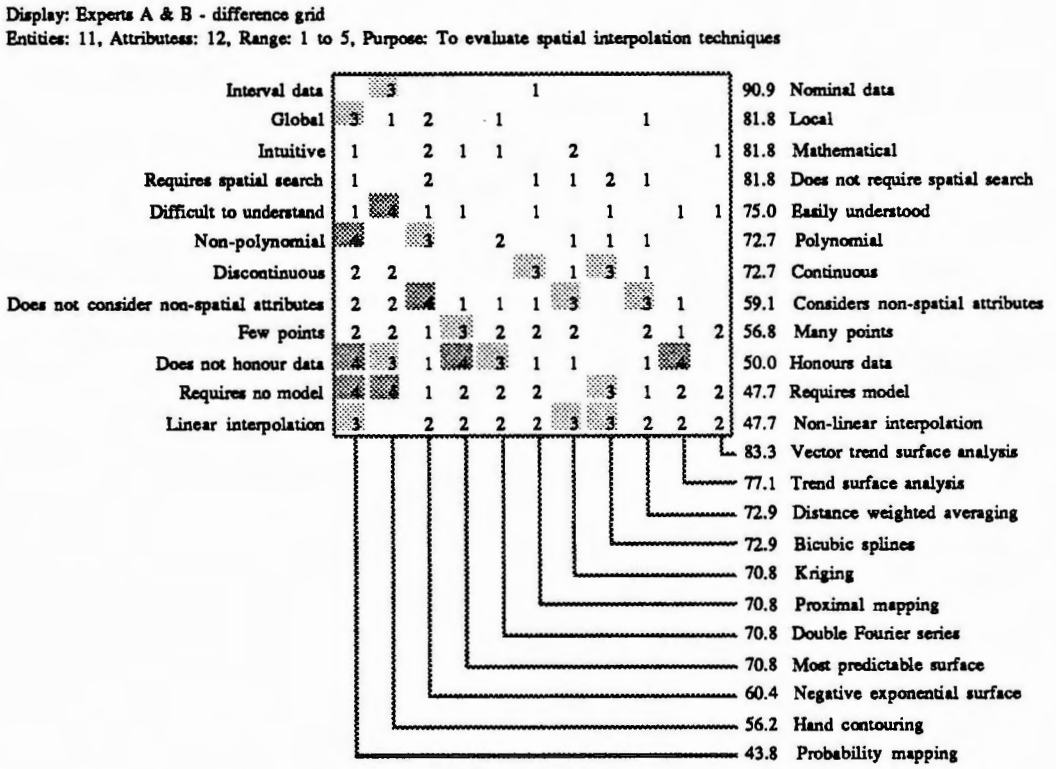


Fig.13 The difference grid for experts A and B

```
G1<:G2 62.5% over 80.0 (ExpertA attribute-construed-by ExpertB)
  1:   6.2% ≥ 88.5    G1A2: Local - Global
                      G2A2: local - global
  2:  12.5% ≥ 87.5    G1A3: Low level data - High level data
                      G2A8:   nominal data - interval or ratio data
  3:          86.5    G1A1: Does not honour data points - Honours data points
                      G2A4:  doesn't honour data points - honours data points
  4:          86.5    G1A7: Short distance autocorrelation - Long distance autocorrelation
                      G2A2:                              local - global
  5:  31.2% ≥ 86.5    G1A9: New geographical technique - Old geographical technique
                     G2A18:              not widely used - widely used
  6:  37.5% ≥ 85.4   G1A16: Not widely used - Widely used
                     G2A18: not widely used - widely used
  7:  43.8% ≥ 83.3    G1A5: Discontinuous - Continuous
                      G2A2:          local - global
  8:  50.0% ≥ 82.3    G1A4: Mathematically complex - Mathematically simple
                     G2A11:   heavy computing load - no computing load
  9:          81.2   G1A10:   Hard to adapt to multivariate - Easy to adapt to multivariate
                      G2A5: usually one variable considered - multiple variables considered
 10:  62.5% ≥ 81.2   G1A12:   Does not require spatial search - Requires spatial search
                     G2A13: estimates susceptible to clusters - not as susceptible to clusters
 11:          79.2    G1A6: Does not require a priori model - Requires a priori model
                      G2A2:                                  local - global
 12:  75.0% ≥ 79.2   G1A11:      Few points - Many points
                     G2A18: not widely used - widely used
 13:          76.0   G1A13:          Does not use polynomial - Uses polynomial
                      G2A6: doesn't fit a mathematical curve - mathematical curve fitting
 14:  87.5% ≥ 76.0   G1A15: Not very effective - Very effective
                     G2A17: not very effective - very effective
 15:  93.8% ≥ 72.9   G1A14: Not very important - Very important
                     G2A18:    not widely used - widely used
 16: 100.0% ≥ 71.9    G1A8:        Models the stationarity - Assumes stationarity
                      G2A3: autocorrelation not considered - autocorrelation considered
```

Fig.14 Attribute comparison of expert B with expert A

Figure 14 shows an attribute comparison from the geographic study in which expert B has specified attributes and filled in the values for a class defined by entities elicited from expert A. The print out shows the matches sorted with best first. The cumulative percentage is given of those with matches greater than the value shown, and the attribute from B which best matches each from A is shown beneath it. It can be seen that:

- The first or highest match which accounts for 6.2% of the attributes has a level of 88.5 out of a possible 100 if they were identical. That is, both experts are using the attribute *local-global* in the same way. This is an example of *correspondence*. However, it is not an interesting one because we can see that the terms used are the same and it is effectively arising from a *consensus*.
- The first and second matches together account for 12.5% of all attributes, and they are matched over the level of 87.5. For the second attribute, when Expert A uses the term *low-level-data—high-level data,* expert B is using the term *nominal data—interval or ratio data.* This shows a difference in terminology which can be interpreted as their *levels of abstraction* being different in their construing of this topic. This illustrates the two experts having attributes in *correspondence*.
- The third match again shows a *correspondence* that can be interpreted as arising from *consensus*, with both experts using the attribute *does not honour data points—honours data points* in the same way. Note that this showed up as *conflict* in Figure 12, probably reflecting that, in this study, the methodology was more complex and the exchange grids from which Figure 12 derives were elicited and discussed *before* those from which Figure 14 derives.

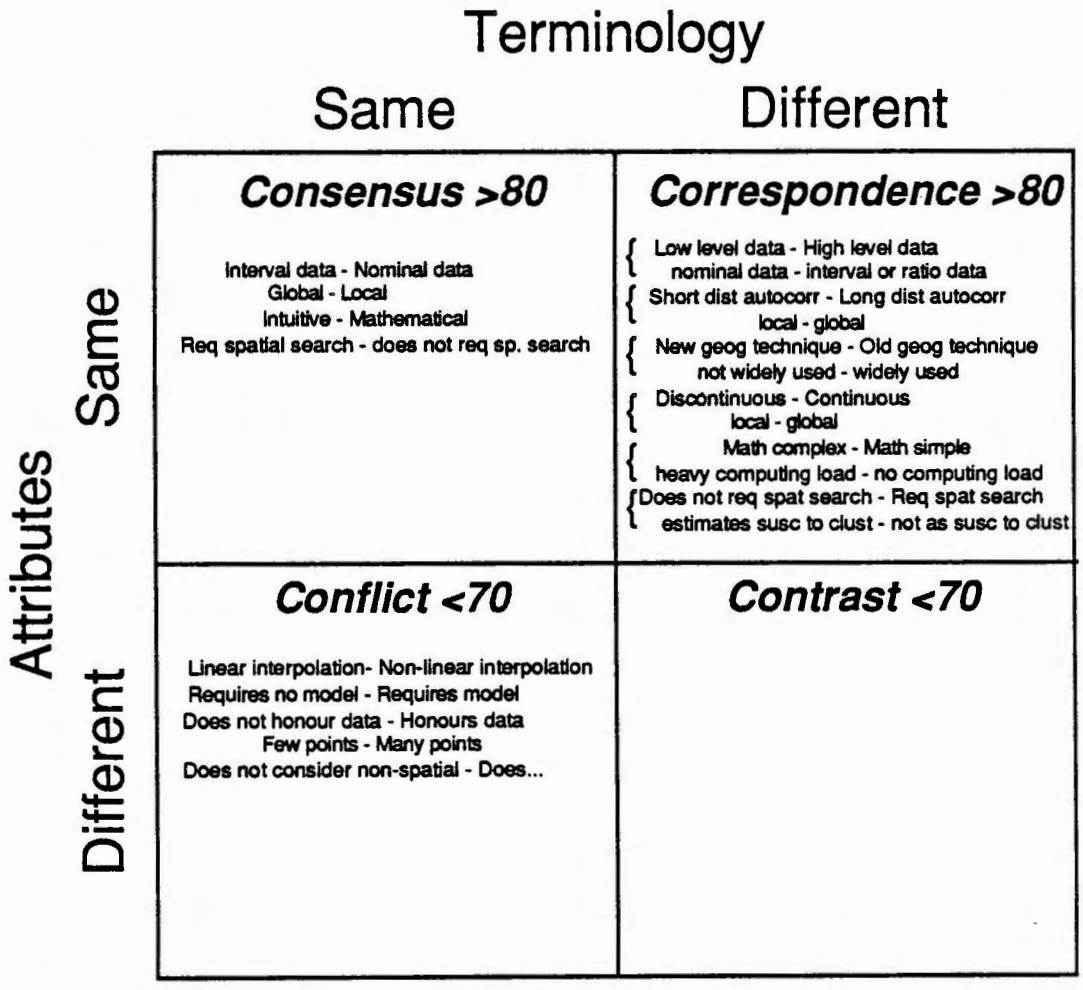


Fig.15 Consensus, correspondence, conflict and contrast from Figs.12 and 14

- The fourth match shows a *correspondence* between *short-distance autocorrelation—long-distance autocorrelation* and *local—global*. Notice that *local—global* used by expert B was also used in the first match indicating that expert A has two attributes *short-distance autocorrelation—long-distance autocorrelation* and *local—global* which are used similarly to each other but with different terminology, whereas expert B has only one. In fact, looking on to the seventh and eleventh matches, it can be seen that expert A has two more attributes *discontinuous—continuous* and *does not require a priori model—requires a priori model* which correspond to expert B's single attribute *local—global*. This shows a differences in richness of concepts not necessarily making new distinctions in the class so far defined by the entities.
- The eighth match, still over the level of 82, again shows *correspondence*. It shows the attribute *heavy computing load—no computing load* is being used by expert B to correspond to *mathematically complex—mathematically simple* used by expert A. We can interpret this as a difference in terminology corresponding to a correlation in the real-world.

Figure 15 shows each of the attributes from Figures 12 and 14 put into the appropriate quadrant of Figure 2. There is no significant example of *contrast* in this data, possibly because the two experts work very closely together. Such examples do arise in a full analysis of the three experts in the original study.

```
Mode Attribute 1: 13 attributes in 3 grids at 80.0
  G1A1:   Does not honour data points - Honours data points
  G1A2:                        Global - Local
  G1A5:                    Continuous - Discontinuous
  G1A7: Long distance autocorrelation - Short distance autocorrelation
  G2A2:                        global - local
  G2A4:    doesn't honour data points - honours data points
  G2A7:                    parametric - nonparametric
  G2A9:       requires periodicities - doesn't require periodicities
 G2A10:      fits a trend to the data - doesn't fit a trend
 G2A12:     assumes isotropic surface - assumes anisotropic surface
  G3A1:                        global - local
  G3A6:                   periodicity - non-periodicity
  G3A7:                   very smooth - non-smooth data

Mode Attribute 2: 8 attributes in 3 grids at 80.0
 G1A12:   Does not require spatial search - Requires spatial search
  G2A3:    autocorrelation not considered - autocorrelation considered
 G2A13: estimates susceptible to clusters - not as susceptible to clusters
 G2A15:                    representative - interpretive
  G3A3:                data restrictions - no data restrictions
  G3A8:                low computer cost - high computer cost
  G3A9:               no error estimates - error estimates
 G3A13:       not so effective technique - very effective technique

Mode Attribute 3: 6 attributes in 3 grids at 80.0
  G1A4:     Mathematically complex - Mathematically simple
  G1A9: New geographical technique - Old geographical technique
 G1A16:           Not widely used - Widely used
 G2A11:      heavy computing load - no computing load
 G2A18:           not widely used - widely used
 G3A11:        non-linear surface - linear surface

Mode Attribute 4: 4 attributes in 3 grids at 80.0
  G1A3: Low level data - High level data
  G2A8:   nominal data - interval or ratio data
  G3A2:        nominal - interval
  G3A5: non-continuous - continuous

Mode Attribute 5: 2 attributes in 2 grids at 80.0
 G1A10:   Hard to adapt to multivariate - Easy to adapt to multivariate
  G2A5: usually one variable considered - multiple variables considered
```

Fig.16 Mode attributes from three experts

Figure 16 shows a *mode attributes* analysis of data from the three geographers which extends the clusters noted above. These mode attributes are essentially inter-conceptual-system clusters of corresponding attributes from the three experts. The first may be interpreted as centering around *Global-Local* encompassing a variety of concepts related to this; the second as around autocorrelation techniques and their consequences; the third as around the complexity of the technique; the fourth around the type of data; and the fifth around the number of variables considered.

These five mode attributes can be interpreted as indicating stereotypical lines of reasoning most used by these experts. This output can be used as a basis for discussion among the experts on whether these conceptual clusters should be split because they confound different concepts expressed in apparently corresponding attributes, or retained as being the same concept expressed in different terminologies. Once this form of analysis has been discussed by the group it is readily edited and extended.

```
Attribute Links
 G3<:G2    71.4% over   80.0 (ExpertC attribute-construed-by ExpertB)
 G1<:G2    62.5% over   80.0 (ExpertA attribute-construed-by ExpertB)
 G2<:G1    61.1% over   80.0 (ExpertB attribute-construed-by ExpertA)
 G2<:G3    44.4% over   80.0 (ExpertB attribute-construed-by ExpertC)
 G3<:G1    42.9% over   80.0 (ExpertC attribute-construed-by ExpertA)
 G1<:G3    31.3% over   80.0 (ExpertA attribute-construed-by ExpertC)
```

Fig.17 Socionet analysis of three experts

Figure 17 shows a *socionets* analysis of the same data based on a set of comparisons like that shown in full in Figure 14. It can be seen from the first two links that the conceptual system of expert B encompasses the majority of attributes used by experts A and C. This indicates that expert B has a deeper knowledge of the topic than either expert A or expert C. That of expert A encompasses the majority of the attributes of B, shown in the third link. However, that of expert C does not encompass many of those of A and B, and that of A does not encompass many of those of C indicating that C has a different point of view, or a background of different experiences from those of A and B. This could then be explored in detail with expert C.

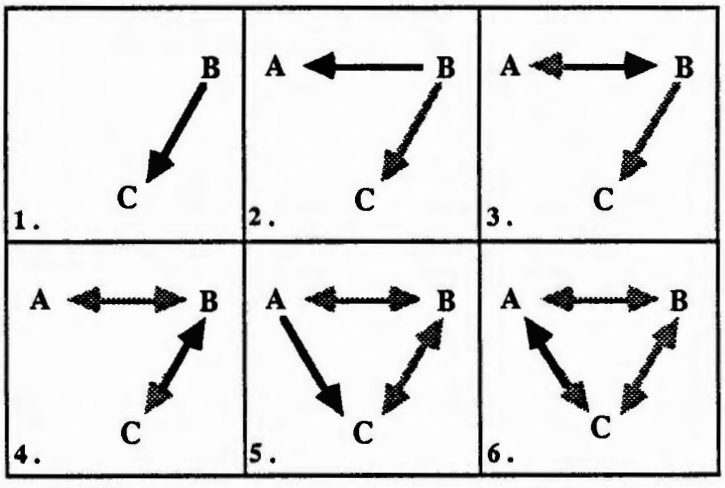


Fig.18 Socionet of relations between experts' conceptual systems

Figure 18 shows this information expressed in the socionet links of relations between the experts' conceptual systems for the domain. Each new link is shown by a black arrow as it is added into the sequence.

SUMMARY OF METHODOLOGY

Phase 1: Domain Discussion and Instantiation

1. The problem is discussed with the experts and a domain identified.
2. The experts are introduced to the use of KSS0. At this time a specific task is agreed upon and a purpose for the grid elicitation developed.
3. The first elicitation consists of the elicitation of a grid by each expert individually. This gives each expert experience in using KSS0 and provides an initial set of entities and attributes particular to each expert.
4. These grids are processed using FOCUS and PrinCom, and the results reported and explained to each expert respectively.
5. The experts meet together to discuss each set of entities, review and revise the purpose or context of the elicitations, and agree on a common set of entities which all understand.

Phase 2: Conceptualization and Feedback

6. This common entity set provides the basis for the next phase of data collection which consists of the elicitation of a grid by each expert using the set of common entities and a common rating scale. This elicitation is done separately for each expert.
7. These grids are also processed on FOCUS and PrinCom, and the results reported and explained to each expert respectively.
8. This data is then processed on Socio, and attributes of correspondence and contrast identified.

Phase 3: Exchange and Compare

8. The next phase of data collection consists of the experts exchanging their first grids (entities and attributes) with each other expert in the group to produce other grids.
10. This data is then processed on Socio, and attributes of consensus and conflict identified.
11. From the Socio results a mode grid is produced to identify the content of the shared lines of thought emanating from the distinctions made by all the experts together.
12. From the Socio results the socionet sequence is produced to identify the subgroups of experts who think and act in similar ways, and any who are different in their thinking from the main lines of thought in the group.

Phase 4: Rule Derivation and Validation

13. This data is then processed using Entail (Gaines & Shaw 1986) to produce rules, classes and/or objects with which to prime an expert system shell.
14. The final data collection phase consists of giving each expert a selected list of entailments generated from this grid. A set of examples from four separate levels of significance of entailment are then selected and randomly presented to the expert who is asked to rate each rule on a four point scale from correct to incorrect. This data is used to check that the set of rules produced by Entail is an accurate one.
15. The rules are tagged with any information resulting from these analyses, such as the name of the expert, his contribution to the group of experts, and his relationship with the other experts in the group with respect to the sub-domain under consideration.

CONCLUSIONS

The methodology described in this paper provides facilities for revealing the similarities and differences in the concept systems of different experts, or the same experts at different times, construing a domain defined through common entities or attributes. It can be used to focus discussion between experts on those differences between them which require resolution, enabling them to classify them in terms of differing terminologies, levels of abstraction, disagreements, and so on.

Note that the derivation of consensual, corresponding, conflicting and contrasting attributes is completely algorithmic, based solely on the data in the grids. This derivation is done by a computer program, not a knowledge engineer, and its basis can be demonstrated clearly to the experts through computer output such as the difference grid of Figure 13. Thus, there are no opinons being expressed about the correctness of the use of the attributes and terminology, that the differences highlighted are 'right' or 'wrong.' It is open to the experts to consider, discuss and explain these differences, changing or retaining them as they wish. Conflicts can be retained in the final system if desired by tagging classes, objects and rules with the sources from which they derive.

Note also that the methodology described applies equally to the relations between the conceptual structures of experts and representative clients for the system. The differences in conceptual systems shown in Figure 3 and the problems that may arise from them may be made overt, and the methods for overcoming these for the different classes of clients, may be analyzed in detail using the derivation of consensual, corresponding, conflicting and contrasting attributes from experts and clients.

The methodology promotes the full exploration of the conceptual framework of a domain of expertise by encouraging experts to operate in a "brain-storming" mode as a group, using differing viewpoints to develop a rich framework. It avoids social pressures forcing an invalid consensus by providing objective analysis of separately elicited conceptual systems.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. The KSS0 system was made available by the Centre for Person Computer Studies. We are grateful to many colleagues at the knowledge acquisition workshops for discussions which have highlighted many of the issues raised in this paper.

REFERENCES

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. & 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.

Diederich, J., Ruhmann, I. & May, M. (1987) KRITON: a knowledge-acquisition tool for expert systems. **International Journal of Man-Machine Studies(26)**, 29-40.

Gaines, B.R. (1987) Rapid prototyping for expert systems. Oliff, M. (Ed). **Proceedings of International Conference on Expert Systems and the Leading Edge in Productions, Planning and Control.** pp. 213-241. University of South Carolina.

Gaines, B.R. & Shaw, M.L.G. (1986). Induction of inference rules for expert systems. **Fuzzy Sets and Systems**, 8(3), 315-328 (April).

Hayes-Roth, F. (1984). The industrialization of knowledge engineering. Reitman, W., Ed. **Artificial Intelligence Applications for Business**. pp. 159-177. Norwood, New Jersey: Ablex.

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

Mancuso, J.C. & Shaw, M.L.G., Eds. (1988). **Cognition and Personal Structure: Computer Access and Analysis**. New York: Praeger Press.

Shaw, M.L.G. (1980). **On Becoming a Personal Scientist**. London: Academic Press.

Shaw, M.L.G., Ed. (1981). **Recent Advances in Personal Construct Technology**. London: Academic Press.

Shaw, M.L.G. (1985). Communities of knowledge. Epting, F. & Landfield, A.W., Eds. **Anticipating Personal Construct Psychology**. pp. 25-35. Lincoln: University of Nebraska Press.

Shaw, M.L.G. (1988). An interactive knowledge-based system for group problem-solving. **IEEE System, Man and Cybernetics**, (in press).

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. & Woodward, J.B. (1987) Validation of a knowledge support system. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.18-0-18-15. Banff (October).

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

RECENT PROGRESS IN AQUINAS: A KNOWLEDGE ACQUISITION WORKBENCH

John H. Boose, David B. Shema, Jeffrey M. Bradshaw

Boeing Advanced Technology Center, 7L-64
Boeing Computer Services
P.O. Box 24346, Seattle, WA 98124

ABSTRACT

Acquiring knowledge from a human expert is a major problem when building a knowledge-based system. AQUINAS, an expanded version of the Expertise Transfer System (ETS), is a knowledge acquisition workbench that combines ideas from psychology and knowledge-based systems research to support knowledge acquisition tasks. AQUINAS interviews experts directly and helps them organize, analyze, test, and refine their knowledge bases. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined, giving consensus and dissenting opinions among groups of experts. Results from user consultations are derived from information propagated through hierarchies. ETS and AQUINAS have assisted in building knowledge based systems for several years at The Boeing Company

This paper describes recent progress on AQUINAS in the areas of knowledge base performance measurement, knowledge base maintenance, interacting trait constraints, consultation graphics, and eliciting strategic and procedural knowledge Experiments show how AQUINAS can automatically improve knowledge bases and even suggest new problem-solving information. Forms of interactive and automatic machine learning employed by AQUINAS are also discussed.

KNOWLEDGE-BASED SYSTEMS AND RAPID PROTOTYPING

Knowledge based systems provide environments where preliminary knowledge bases from human experts may be rapidly prototyped in a matter of days or weeks. Experts, end-users, and project managers may easily examine the system's behavior and suggest changes that can be rapidly implemented. The knowledge, often stored in english-like rules, is readily available for examination, discussion, and modification by all project participants. The rapid prototyping capability of these environments is one reason why they have been successful. One of the first of these environments, EMYCIN (Buchanan and Shortliffe, 1985) containing the knowledge transfer tool TEIRESIAS (Davis and Lenat, 1982, is still one of the best examples of such a rapid prototyping environment. TEIRESIAS helped knowledge engineers and experts enter rules and parameters, checked consistency, debugged consultations, and even suggested new rules The interface and rule language could be used by the expert with a relatively small amount of training.

The major hurdle when building these systems is that of acquiring and modeling the human's expertise. A knowledge engineer - someone familiar with the knowledge-based system environment or "shell" - interviews the expert, models the knowledge, embeds it in the system, and reviews the system's behavior with the expert. The knowledge engineer changes the system based on the suggestions of the expert. Cycles of revision and review can take from six to twenty-four months before the system exhibits expertise when using manual interviewing methods, even in a narrow domain.

Automated knowledge acquisition tools are being developed to help cut down the revise-and-review cycle time. These tools interview experts directly and help them refine, structure, and test their knowledge. Expertise from many experts may be rapidly combined together and used Because models can be built rapidly and easily revised, AQUINAS can be effectively used as a general-purpose decision aid as well as a tool for knowledge engineering

AQUINAS SYSTEM DESCRIPTION

This section discusses AQUINAS' knowledge representation and basis in Personal Construct Theory. Knowledge analysis and refinement tools, and the system's rapid consultation prototyping capability will also be demonstrated.

AQUINAS, an expanded version of the Expertise Transfer System (ETS; Boose, 1984, 1985, 1986a), is a knowledge acquisition workbench that combines ideas from psychology and knowledge-based systems research to support knowledge acquisition tasks These tasks include eliciting distinctions, decomposing problems, combining uncertain information, incremental testing, integration of data types, automatic expansion and refinement of the knowledge base, use of multiple sources of knowledge, and providing process guidance (Boose and Bradshaw, 1987). AQUINAS interviews experts and helps them analyze, test, and refine the knowledge base. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined. Results from user consultations are derived from information propagated through hierarchies. AQUINAS delivers knowledge by creating knowledge bases for several different expert system shells. Help is given to the expert by a dialog manager that embodies knowledge acquisition heuristics.

Using AQUINAS, rapid prototypes of knowledge-based systems can be built in as little as one hour, even when the expert has little understanding of knowledge-based systems or has no prior training in the use of the tool. The interviewing methods in AQUINAS are derived from George Kelly's Personal Construct Psychology and related work (Kelly, 1955; Shaw and Gaines, 1987; Boose, 1988). Kelly's methods and theory provide a rich framework for modeling the qualitative and quantitative distinctions inherent in an expert's problem-solving knowledge.

AQUINAS tools mentioned in this paper are explained more fully elsewhere (Boose, 1988; Boose and Bradshaw, 1987; Kitto and Boose, 1988, Shema and Boose, 1988). AQUINAS is written in Interlisp and runs on the Xerox family of Lisp machines. Subsets of AQUINAS also run in an Interlisp version on the DEC Vax, SUN III and IV, and a micro computer "C/UNIX"-based portable version that runs on a variety of workstations.

Knowledge Represented in Repertory Grids

The expert enters and refines knowledge in the form of repertory grids. In a repertory grid, problem solutions - ***elements*** - are elicited and placed across the grid as column labels, and traits of these solutions - ***constructs*** - are listed alongside the rows of the grid (Figure 1). Traits are first elicited by presenting groups of solutions and asking the expert to discriminate among them. Following this, the expert gives each solution a rating showing where it falls on the trait scale, and gives the relative importance of each trait. Below are excerpts from an expert building a repertory grid for a heart dysrhythmias portion of a medical aid advisor.

First, the expert partitions the medical aid advisor into several sub-problems, or *cases*. Then, for the HEART.DYSRHYTHMIAS case, the expert enters a list of possible problems (termed *solutions* in AQUINAS; the goal of this part of the system is to diagnose the problem):

```
:
Please enter a list of items for expert DBS that are potential solutions for HEART.DYSRHYTHMIAS. Enter them one to a
line. When you're done, enter a RETURN.
NEW SOLUTION** NORMAL SINUS RHYTHM
NEW SOLUTION** SINUS ARRHYTHMIA
NEW SOLUTION** SINUS TACHYCARDIA
NEW SOLUTION** SUPRAVENTRICULAR TACHYCARDIA
NEW SOLUTION** SINUS BRADYCARDIA
NEW SOLUTION** ATRIAL FLUTTER
NEW SOLUTION** ATRIAL FIBRILLATION
NEW SOLUTION** NODAL RHYTHM
NEW SOLUTION** 1ST DEG HEART BLOCK
NEW SOLUTION** 2ND DEG HEART BLOCK-WENCKEBACH
NEW SOLUTION** 2ND DEG HEART BLOCK-MOBITZ II
NEW SOLUTION** 3RD DEG HEART BLOCK
NEW SOLUTION** VENTRICULAR TACHYCARDIA
NEW SOLUTION** VENTRICULAR FIBRILLATION
NEW SOLUTION** IDIOVENTRICULAR RHYTHM
NEW SOLUTION** ASYSTOLE
```

AQUINAS asks the expert for traits that help distinguish between sets of solutions.

```
Think of an important new trait that two of NORMAL.SINUS.RHYTHM, SINUS.ARRHYTHMIA, and SINUS.TACHYCARDIA
share, but that the other one does not. What is that trait?
NEW TRAIT (EXTREME)** REGULAR.RHYTHM

What is that trait's opposite as it applies in this case?
NEW TRAIT (OPPOSITE)** IRREGULAR.RHYTHM

What is the name of a scale or concept that describes REGULAR.RHYTHM / IRREGULAR.RHYTHM?
NEW TRAIT (CONCEPT)** HEART.RHYTHM

Think of an important new characteristic that two of SINUS.ARRHYTHMIA, SINUS.TACHYCARDIA, and
SUPRAVENTRICULAR.TACHYCARDIA share, but that the other one does not. What is that characteristic?
NEW TRAIT (EXTREME)** SLOW.HEART.RATE

What is that characteristic's opposite as it applies in this case?
NEW TRAIT (OPPOSITE)** FAST.HEART.RATE

What is the name of a scale or concept that describes SLOW.HEART.RATE / FAST.HEART.RATE?
NEW TRAIT (CONCEPT)** HEART.RATE

Think of an important new attribute that two of SINUS.TACHYCARDIA, SUPRAVENTRICULAR.TACHYCARDIA, and
SINUS.BRADYCARDIA share, but that the other one does not. What is that attribute?
NEW TRAIT (EXTREME)** NORMAL.P.WAVE

What is that attribute's opposite as it applies in this case?
NEW TRAIT (OPPOSITE)** ABNORMAL.P.WAVE

What is the name of a scale or concept that describes NORMAL.P.WAVE / ABNORMAL.P.WAVE?
NEW TRAIT (CONCEPT)** P.WAVE.CHARACTERISTIC
   :
```

The expert then rates each solution on each trait. By default, AQUINAS makes each trait into an ordinal trait with rating values from 1 to 5. Later, the expert can change the trait type to nominal, interval, or ratio, and change trait value ranges and intervals. AQUINAS can assist with this in certain situations, and even suggest new trait types.

```
Rate these solutions on the concept HEART.RHYTHM using a scale of 1 to 5, where 1 means more like REGULAR.RHYTHM
and 5 means more like IRREGULAR.RHYTHM. If neither one seems to apply, enter NONE. If both seem to apply, enter
BOTH. If you would like help estimating values, enter USE.BARCHART. If you wish to change the type of the trait, enter
CHANGE.
 REGULAR.RHYTHM(1)  IRREGULAR.RHYTHM(5)
  NORMAL.SINUS.RHYTHM** 1
  SINUS.ARRHYTHMIA** 4
  SINUS.TACHYCARDIA** 1
  SUPRAVENTRICULAR.TACHYCARDIA** 1
  SINUS.BRADYCARDIA** 1
  ATRIAL.FLUTTER** 2
  ATRIAL.FIBRILLATION** 5
  NODAL.RHYTHM** 1
  1ST.DEG.HEART.BLOCK** 1
  2ND.DEG.HEART.BLOCK-WENCKEBACH** 4
  2ND.DEG.HEART.BLOCK-MOBITZ.II** 1
  3RD.DEG.HEART.BLOCK** 1 CF .6 5 CF 4
  VENTRICULAR.TACHYCARDIA** 1
  VENTRICULAR.FIBRILLATION** 5
  IDIOVENTRICULAR.RHYTHM** 5
  ASYSTOLE** 1
   :
```

Note that the expert entered a *rating distribution* for 3RD.DEGREE.HEART.BLOCK. More complex distributions can also be entered graphically. The expert then rates the importance of each trait's contribution to finding a solution.

```
Please rate the relative importance of HEART.RHYTHM (REGULAR.RHYTHM / IRREGULAR.RHYTHM) on a scale from 5 to
0, where 5 means more important, and 0 means less important.
HEART.RHYTHM** 5

Please rate the relative importance of HEART.RATE (SLOW.HEART.RATE / FAST.HEART.RATE) on a scale from 5 to 0,
where 5 means more important, and 0 means less important.
```

HEART.RATE** 5

Please rate the relative importance of P.WAVE.CHARACTERISTIC (NORMAL.P.WAVE / ABNORMAL.P.WAVE) on a scale from 5 to 0, where 5 means more important, and 0 means less important.
P.WAVE.CHARACTERISTIC** 4
:

Repertory Grid Analysis

After an initial grid is constructed, AQUINAS helps the expert refine and expand the knowledge base by invoking a variety of analysis tools *Similarities* between solutions and traits are analyzed to help the expert refine useful distinctions and eliminate those that are inconsequential or redundant. For example, NORMAL.SINUS.RHYTHM and 1ST.DEG.HEART.BLOCK are found to be highly matched, and the expert enters a new trait, PR.INTERVAL, to help further distinguish between them. After the new row is added, AQUINAS asks the expert to rate all the solutions on the new trait.

The two solutions **NORMAL.SINUS.RHYTHM** and **1ST.DEG.HEART.BLOCK** are matched at the **95**% level. Can you think of some new trait that would distinguish between them?
AQU** **YES**

What is the name of that trait?
NEW TRAIT (CONCEPT)** **PR.INTERVAL**
:

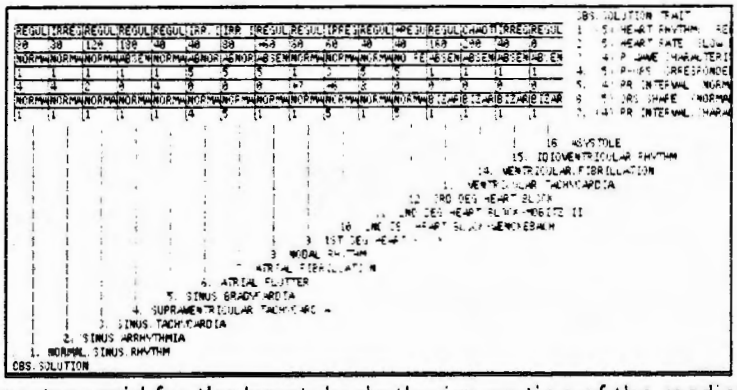


Figure 1. Repertory grid for the heart dysrhythmias portion of the medical aid advisor.

Inductive implications between trait values can show the expert higher levels of abstraction implied by a repertory grid (Figure 2). If the expert disagrees with an implication, AQUINAS helps the expert refine the grid. This is one way in which interactive learning takes place in AQUINAS.

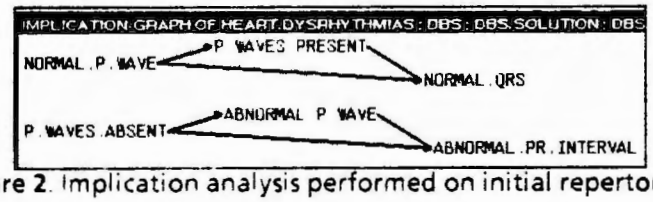


Figure 2. Implication analysis performed on initial repertory grid.

Trait scale types may be changed from the original default (ordinal scale with ranges from one to five) to other types (nominal, interval, ratio scales with user-defined ranges) based on precision needed, cost of elicitation, and representation convenience AQUINAS helps the expert change trait scale types in several ways. Here, AQUINAS notices that one trait has only extreme rating values (all ratings are either 1 or 5), and suggests that the trait scale type be changed from ordinal to nominal The expert agrees, and AQUINAS automatically maps the numerical values to symbolic nominal values.

P.WAVE.CHARACTERISTIC has only extreme values. Should its trait type be changed from ORDINAL to NOMINAL?
AQU** **YES**
Use the default method for mapping old values to new values of P.WAVE.CHARACTERISTIC?
AQU** **YES**

```
Mapping for NORMAL.SINUS.RHYTHM from 1 to:   NORMAL.P.WAVE 1.00 DEFAULT MAPPING
Mapping for SINUS.ARRHYTHMIA from 1 to:   NORMAL.P.WAVE 1.00 DEFAULT MAPPING
   :
```

Nominal, ordinal, and interval trait scales were illustrated in the grid in Figure 1.

Hierarchical Repertory Grids

In a major extension to Kelly's methods, AQUINAS allows experts to arrange repertory grid knowledge in hierarchies and lattices. Hierarchical tools in AQUINAS help the expert build, edit, and analyze knowledge in hierarchies These hierarchies allow the expert to break up complex problems into pieces of convenient size and similar levels of abstraction. Hierarchies in AQUINAS are organized around *solutions*, *traits*, *knowledge sources* (i.e., experts), and *cases* (Figures 3 and 4).

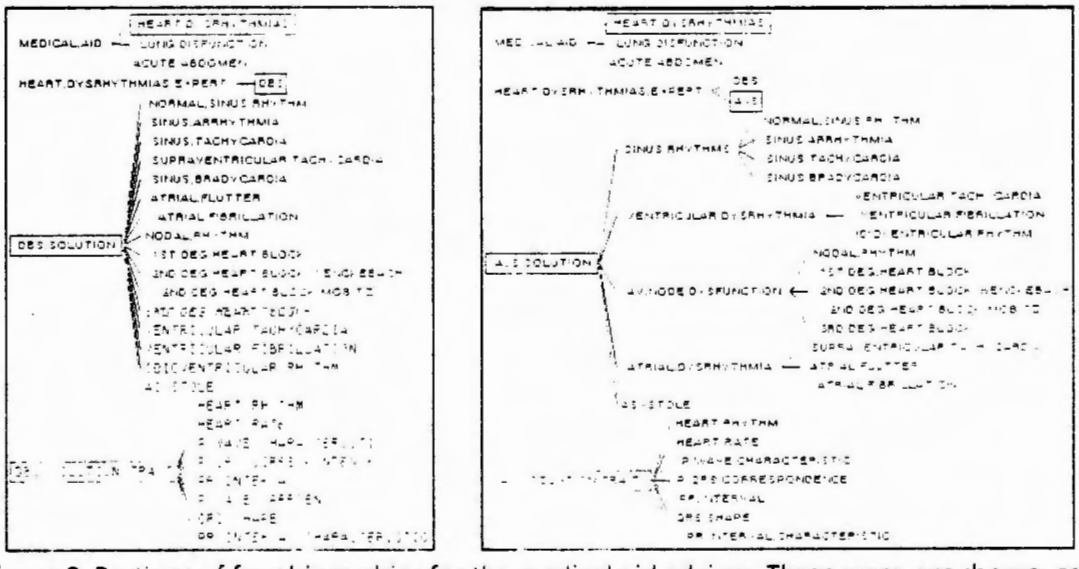


Figure 3. Portions of four hierarchies for the medical aid advisor. Three cases are shown, as well as expert DBS s solution and trait hierarchies. So far, DBS is the only one who has entered information about heart dysrhythmias

Figure 4. Another expert, AJS, has entered knowledge for the same case Tools in AQUINAS were used to help AJS structure solutions in a class hierarchy. The hierarchies are related from top to bottom: given the HEART.DYSRHYTHMIAS case, experts DBS and AJS exist (other experts could exist for other cases). Given expert AJS, we see his solution hierarchy (the top of the hierarchy is the class AJS.SOLUTION). Given that we are looking at the top-level grid in the hierarchy (the children of AJS.SOLUTION), the immediate children of AJS.SOLUTION.TRAIT apply. Different traits can apply to different class levels in the solution hierarchy; different solutions could come from different experts, and so on.

Nodes in the four hierarchies combine to form repertory grids. In the most simple case, the children of a node in a solution hierarchy supply the solutions along the top of a grid; the children of a node in a trait hierarchy supply the traits down the side of a grid (Figure 4). Rating values within the grid are judgments about the solutions with respect to each trait (Figure 5).

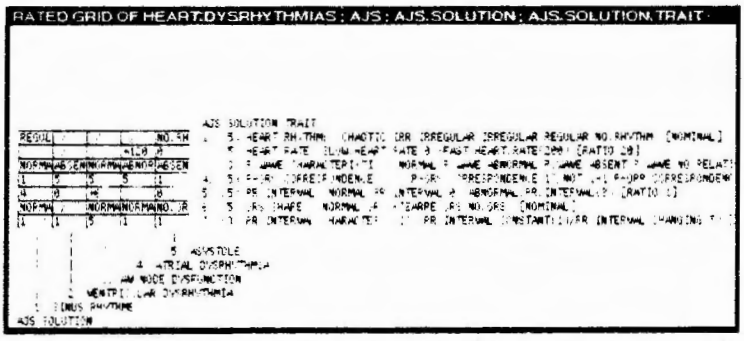


Figure 5. AJS's top-level grid shows *solution classes* and traits. Separate grids exist for each class. During consultations, AQUINAS will search the hierarchies to ask questions and infer ratings.

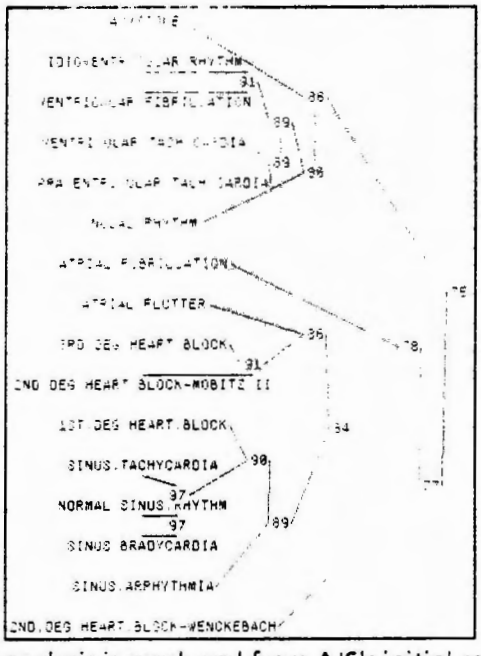


Figure 6. A solution cluster analysis is produced from AJS's initial repertory grid. AJS is asked to form hierarchies by labeling junctions on the cluster. The clusters SINUS.RHYTHMS, VENTRICULAR.DYSRHYTHMIA, AV.NODE.DYSFUNCTION, and ATRIAL.DYSRHYTHMIA, shown in Figure 4, are formed.

Strategies for helping the expert build and refine hierarchies in AQUINAS include *laddering* (eliciting specializations and generalizations through tailored dialogs), *cluster analysis* (experts are asked to label levels of abstraction represented by cluster junctions: see Figure 6), *implication pattern analysis* (certain patterns may suggest levels of hierarchical subsumption), and *trait value examination* (certain combinations of trait values may suggest that hierarchical expansion is appropriate).

Decisions and Performance: Consultations in AQUINAS

The knowledge embedded in hierarchical repertory grids in AQUINAS can be applied to specific problems by running consultations. During a consultation, AQUINAS asks the user to specify the preferred or absolute values of traits associated with an ideal solution. Consultation results are displayed as rank-ordered, interval lists of solutions.

The model of problem-solving used in AQUINAS is that of multiple knowledge sources (experts) that work together in a common problem solving context (case) by selecting the best alternatives for each of a sequential set of decisions (solutions). Alternatives at each step are selected by combining relevant information about preferences (relativistic reasoning), constraints (absolute reasoning) and evidence (probabilistic reasoning). This paradigm follows one suggested by Clancey who suggested that many problems are solved by abstracting data, heuristically mapping higher level problem descriptions onto solution models, and then refining these models until specific solutions are found (Clancey, 1986). A maximum entropy approach is applied by AQUINAS's reasoning mechanism (Boose and Bradshaw, 1987). ETS used a reasoning approach based on rules and certainty factors (Boose, 1986a).

Knowledge from multiple experts may be rapidly combined using AQUINAS. Users may receive dissenting as well as consensus opinions from groups of experts, thus getting a full range of possible solutions. Disagreement between the consensus and the dissenting opinion can be measured to derive a degree of conflict for a particular consultation. The system can be used for cost-effective group data gathering (Boose, 1986b, 1988).

Following are portions of a transcript from an AQUINAS consultation using expertise from both DBS and AJS. The user specifies trait preferences, and AQUINAS combines the preferences together producing consensus and dissenting opinions:

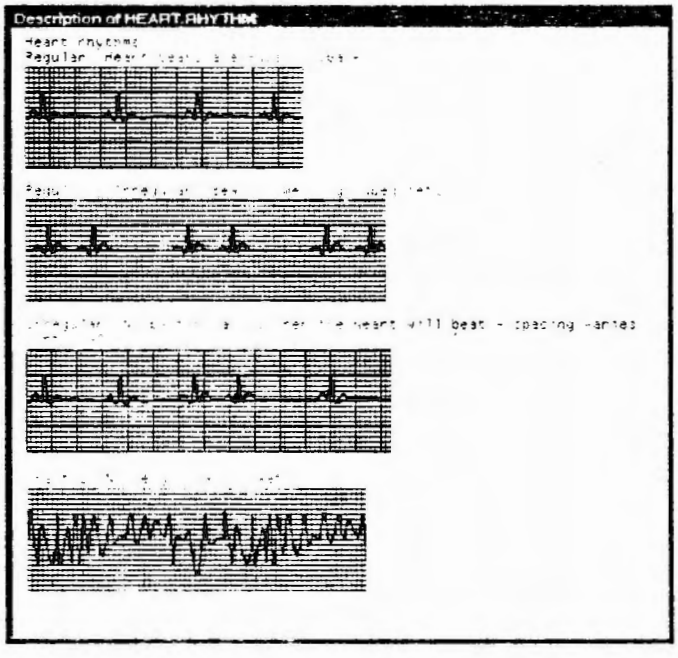
```
:
Enter a brief description of this consultation
AQU** 56 YO MALE FOUND DOWN, UNKNOWN DOWNTIME, NO CPR @ ARRIVAL

What solutions would you like to consider for consultation MEDICAL-AID--ALLEN ? Enter them one to a line. If you wish
every solution to be considered, enter ALL. You may specify solution weights by appending selections with a number
between 0 and 1 (e.g., .25). When done, enter RETURN.
AQU** ALL

The following experts know about the solutions : DBS AJS
Would you like to weight these experts?
AQU** NO

What kind of ordering should be used for questions (COST-BENEFIT SEQUENTIAL USER-SPECIFIED)?
AQU** SEQUENTIAL
:
```

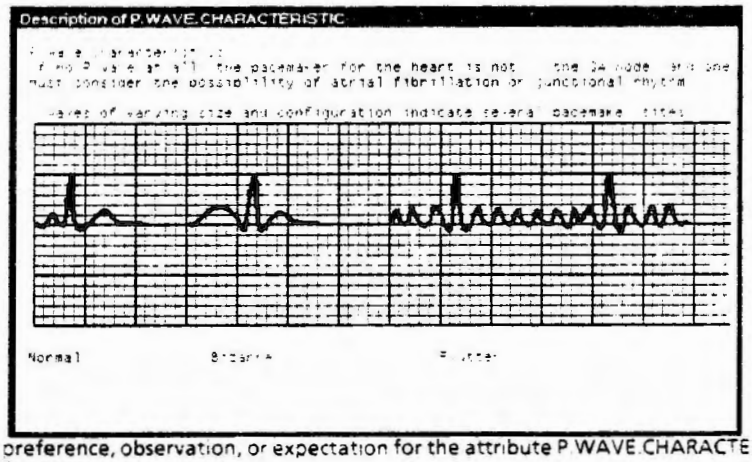
AQUINAS can display graphic information when asking questions. Here, the user answers a consultation question by comparing a scope reading with categories of heart rhythm.



Please indicate your preference, observation, or expectation for the attribute HEART.RHYTHM (AJS). You may also USE.BARCHART for entry.
(CHAOTIC IRR.IRREGULAR IRREGULAR REGULAR NO RHYTHM) WT 5
AQU** **REGULAR**

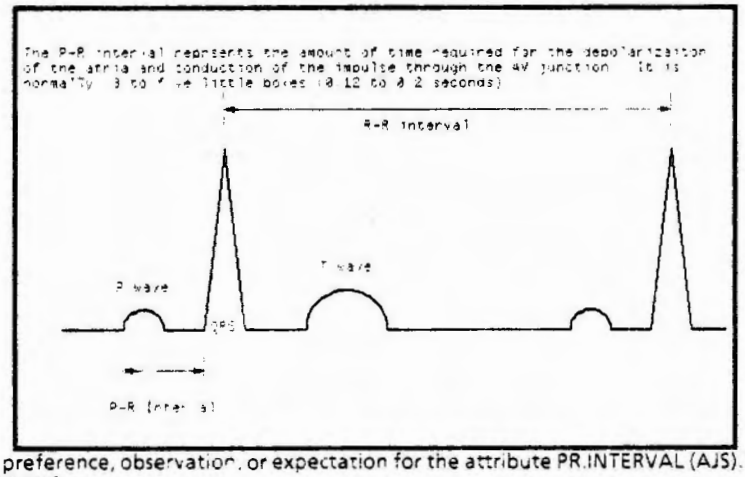
Please indicate your preference, observation, or expectation for the attribute HEART.RATE (AJS).The interval is 20 You may also USE.BARCHART for entry
SLOW.HEART.RATE(0) / FAST.HEART.RATE(200) WT 5
AQU** **60**

Again, the user enters information by comparing a scope reading with graphic information displayed by AQUINAS. The expert or knowledge engineer enters this information by preparing a graphic file that is associated with a trait. Graphics can be created with a variety of paint programs and bitmap editors, or scanned and digitized. Six graphic displays are associated with this knowledge base; they took approximately two hours to enter.



Please indicate your preference, observation, or expectation for the attribute P.WAVE.CHARACTERISTIC (AJS) You may also USE.BARCHART for entry.
(NORMAL.P.WAVE ABNORMAL.P WAVE ABSENT P WAVE NO.RELATION) WT 3
AQU** **NORMAL.P.WAVE**

Please indicate your preference, observation, or expectation for the attribute P-QRS.CORRESPONDENCE (AJS) You may also USE.BARCHART for entry
1-1.P-QRS.CORRESPONDENCE(1) / NOT 1-1.P-QPR.CORRESPONDENCE(5) WT 5
AQU** 1



Please indicate your preference, observation, or expectation for the attribute PR.INTERVAL (AJS).The interval is 1 You may also USE.BARCHART for entry
NORMAL.PR.INTERVAL(0) / ABNORMAL.PR.INTERVAL(8) WT 5
AQU** 4

After the observations are entered, AQUINAS displays the results of the consultation. Weighted contributions from each expert are shown. When only one expert contributes to a solution, the weight is shown at 1.00 (for example, SINUS.RHYTHMS). When more than one expert contributes to a solution each expert's contribution weight is shown. In SINUS.BRADYCARDIA for example DBS and AJS each contribute equally (.5) so a weighted average of their contributions is computed as the total weight for the solution. If the user had weighted the experts differently, this would be taken into account when computing the average.

```
Test results for consultation
MEDICAL-AID--ALLEN

1 : SINUS.RHYTHMS ( 0.98 )                    (AJS 1.00 0.98)
2 : NORMAL.SINUS.RHYTHM ( 0.91 )              (DBS 0.50 0.83)(AJS 0.50 0.99)
3 : SINUS.BRADYCARDIA ( 0.90 )                (DBS 0.50 0.83)(AJS 0.50 0.98)
4 : SINUS.TACHYCARDIA ( 0.87 )                (DBS 0.50 0.77)(AJS 0.50 0.96)
5 : SINUS.ARRHYTHMIA ( 0.83 )                 (DBS 0.50 0.83)(AJS 0.50 0.83)
6 : 1ST.DEG.HEART.BLOCK ( 0.73 )              (DBS 0.50 0.78)(AJS 0.50 0.68)
7 : 2ND.DEG.HEART.BLOCK-MOBITZ II ( 0.64 )    (DBS 0.50 0.61)(AJS 0.50 0.68)
8 : AV.NODE.DYSFUNCTION ( 0.62 )              (AJS 1.00 0.62)
9 : SUPRAVENTRICULAR.TACHYCARDIA ( 0.59 )     (DBS 0.50 0.56)(AJS 0.50 0.62)
10 : 2ND.DEG.HEART.BLOCK-WENCKEBACH ( 0.57 )  (DBS 0.50 0.61)(AJS 0.50 0.52)
11 : NODAL.RHYTHM ( 0.56 )                    (DBS 0.50 0.48)(AJS 0.50 0.64)
12 : ATRIAL.DYSRHYTHMIA ( 0.56 )              (AJS 1.00 0.56)
13 : ATRIAL.FLUTTER ( 0.50 )                  (DBS 0.50 0.39)(AJS 0.50 0.60)
14 : 3RD DEG.HEART.BLOCK ( 0.48 )             (DBS 0.50 0.36)(AJS 0.50 0.59)
15 : VENTRICULAR.FIBRILLATION ( 0.46 )        (DBS 0.50 0.55)(AJS 0.50 0.38)
16 : ASYSTOLE ( 0.45 )                        (DBS 0.50 0.45)(AJS 0.50 0.45)
17 : IDIOVENTRICULAR.RHYTHM ( 0.43 )          (DBS 0.50 0.48)(AJS 0.50 0.38)
18 : ATRIAL FIBRILLATION ( 0.40 )             (DBS 0.50 0.36)(AJS 0.50 0.44)
19 : VENTRICULAR TACHYCARDIA ( 0.40 )         (DBS 0.50 0.42)(AJS 0.50 0.38)
20 : VENTRICULAR.DYSRHYTHMIA ( 0.38 )         (AJS 1.00 0.38)
```

Next, a dissenting opinion is presented. Correlation scores for all experts are listed. A score of 1.0 would represent perfect ordering agreement (lists in the same order); a score of -1.0 would represent perfect disagreement (lists in opposite order). A score of 0.0 would mean that the lists are ordered randomly with respect to each other.

Results from the expert with the "worst" score (the lowest) are presented side-by-side with the consensus opinion appearing above. Only those solutions that the dissenting expert knows about are listed in the comparative consensus. The order of consensus is preserved. In this case, both experts exhibited moderately strong agreement with the consensus.

Users can get an idea of the overall agreement or disagreement among experts by comparing their correlation scores. Users can also see the entire range of acceptable solutions by comparing the consensus and dissenting opinions. In the results below, the first several solutions are in substantially the same order. The user gains more confidence that these may be valid recommendations. If there are wide differences between experts, the user can employ other tools in AQUINAS to compare the appropriate repertory grids to see where the experts' opinions differed.

```
Dissenting opinion for consultation  MEDICAL-AID--ALLEN

Correlation scores for all experts:
  DBS 0.86
  AJS 0.84

AJS has the most dissenting opinion ( 0.84 correlation score)
```

AJS	/ Consensus
1 : NORMAL.SINUS.RHYTHM 0.99	/ SINUS.RHYTHMS 0.98
2 : SINUS.RHYTHMS 0.98	/ NORMAL.SINUS.RHYTHM 0.91
3 : SINUS.BRADYCARDIA 0.98	/ SINUS.BRADYCARDIA 0.90
4 : SINUS.TACHYCARDIA 0.96	/ SINUS.TACHYCARDIA 0.87
5 : SINUS.ARRHYTHMIA 0.83	/ SINUS.ARRHYTHMIA 0.83
6 : 1ST.DEG.HEART.BLOCK 0.68	/ 1ST.DEG.HEART.BLOCK 0.73
7 : 2ND.DEG.HEART.BLOCK-MOBITZ.II 0.68	/ 2ND.DEG.HEART.BLOCK-MOBITZ.II 0.64
8 : NODAL.RHYTHM 0.64	/ AV.NODE.DYSFUNCTION 0.62
9 : SUPRAVENTRICULAR.TACHYCARDIA 0.62	/ SUPRAVENTRICULAR.TACHYCARDIA 0.59
10 : AV.NODE.DYSFUNCTION 0.62	/ 2ND.DEG.HEART.BLOCK-WENCKEBACH 0.57
11 : ATRIAL.FLUTTER 0.60	/ NODAL.RHYTHM 0.56
12 : 3RD.DEG.HEART.BLOCK 0.59	/ ATRIAL.DYSRHYTHMIA 0.56
13 : ATRIAL.DYSRHYTHMIA 0.56	/ ATRIAL.FLUTTER 0.50
14 : 2ND.DEG.HEART.BLOCK-WENCKEBACH 0.52	/ 3RD.DEG.HEART.BLOCK 0.48
15 : ASYSTOLE 0.45	/ VENTRICULAR.FIBRILLATION 0.46
16 : ATRIAL.FIBRILLATION 0.44	/ ASYSTOLE 0.45
17 : VENTRICULAR.DYSRHYTHMIA 0.38	/ IDIOVENTRICULAR.RHYTHM 0.43
18 : VENTRICULAR.TACHYCARDIA 0.38	/ ATRIAL.FIBRILLATION 0.40
19 : IDIOVENTRICULAR.RHYTHM 0.38	/ VENTRICULAR.TACHYCARDIA 0.40
20 : VENTRICULAR.FIBRILLATION 0.38	/ VENTRICULAR.DYSRHYTHMIA 0.38

Constraints

In AQUINAS, *constraints* are interactions between traits specified by the expert or by the end user. Use of constraints allows the expert to show relationships between traits and allows users to express absolute solution expectations as well as preferences.

Experts can enter relationships that constrain traits given the values of other traits. These types of constraints can limit the solution set, prune questions, and affect question ordering during consultations. For example, the following response triggered an expert-supplied constraint. If the heart rate is 40 or below, certain rhythms won't occur. If this happens during a consultation, these rhythms are pruned from the solution candidates (their scores are set to zero and not changed) The expert also entered the constraint that the rhythm asystole is eliminated for all heart rates greater than zero. In another example, the expert entered a constraint that prunes traits during consultations. In effect, this stops certain questions from being asked. For instance, if the P.WAVE.CHARACTERISTIC is ABSENT.P.WAVE, then questions about PR.INTERVAL and PR.INTERVAL.CHARACTERISTIC are not asked. These types of constraints entered by the expert can also affect question ordering.

The user can also enter constraints during a consultation. Here, for example, the user constrains the solution set to those with heart rates greater than 120 by using the keyword ABSOLUTE. Any solutions associated with heart rates less than 120 will be pruned:

```
Please indicate your preference, observation, or expectation for the attribute HEART.RATE (DBS).The interval is 20  You
may also USE.BARCHART for entry.
 SLOW.HEART.RATE(0) / FAST.HEART.RATE(200) WT 5
AQU** 120 > ABSOLUTE
```

The user can also enter constraints for single values:

```
Please indicate your preference, observation, or expectation for the attribute HEART.RHYTHM (DBS).  You may also
USE.BARCHART for entry.
 (REGULAR IRREGULAR IRR.IRREGULAR CHAOTIC) WT 5
AQU** REGULAR ABSOLUTE
```

The constraints established by the user, HEART.RATE greater than 120 and a HEART.RHYTHM of regular, eliminate most of the solutions (all other solutions received scores of zero):

```
Test results for consultation  MEDICAL-AID--EXAMPLE2

1 : SINUS TACHYCARDIA ( .99 )
2 : SUPRAVENTRICULAR.TACHYCARDIA ( 0.83 )
3 : VENTRICULAR.TACHYCARDIA ( 0.69 )
```

The addition of constraints to repertory grids has allowed AQUINAS to represent, in general, any type of information that could be represented in rules. Constraints will be valuable as we attempt to apply AQUINAS to configuration, design, and planning problems (see Discussion, below).

DEBUGGING, MAINTENANCE, VALIDATION AND VERIFICATION

A general problem when modifying knowledge bases is that changes may degrade system performance. This is especially a problem when the knowledge base is large; it may be unclear how changing one item in a knowledge base containing thousands of items will affect overall system performance. A set of consultation review tools in AQUINAS helps alleviate this problem.

Tools for analyzing consultations to improve performance are an important part of the workbench (Shema and Boose, 1988). These tools may modify the knowledge base after checking that existing test cases still give the intended results. Synthesis of traits and test cases help with the validation of the knowledge base. Experiments are underway to measure the utility of these tools.

Case Storage and Playback

Information entered by the expert during a consultation may be stored for later playback. The recording and playback facilities test new changes to the knowledge base using old consultations. The description of the case, the values of the traits, and the expert's expectations of the ranking are recorded with each stored case.

Cases can be run in batch mode, where changes made to the knowledge base to improve the performance of one test case may be automatically tested on all the previous cases. The expert can see whether or not new changes result in overall improvement.

Knowledge Base Changes

Changes that lead to knowledge base performance improvement (or degradation) take several forms:

Changing ratings in a grid. Judgments about the relationships between solutions and traits may be revised.

Changing a trait's weight. A trait's influence may be increased or decreased by changing its weight. Traits can be temporarily ignored by assigning them a weight of zero.

Adding or deleting a trait. Traits that affect the outcome may be added, deleted, or moved to different levels in the hierarchies. New traits are rated against existing solutions and assigned weights.

Adding or deleting a solution. The potential set of solutions that affect the outcome may be expanded or restricted. New solutions are rated against existing traits.

Changing the expectations of outcomes. The expert may revise the expectation of the system's behavior.

Re-organizing existing knowledge. Knowledge can be reorganized to change its affect on performance. For instance, knowledge can be regrouped under different cases in the knowledge bases hierarchies so that it is applied under different circumstances. Experts may also change their ideas about the range of applicability of portions of the knowledge base.

Review mechanisms fall into several categories - *manual*, *directed*, *semi-automated*, and *automated*.

Manual Methods - Inspect Results and Revise

A variety of methods exist to manually inspect consultation results. The expert may review the results and change the knowledge base directly to try and improve the system's performance.

The rank ordering of the solutions may be displayed in list form (as above), tables, bar charts, or pie charts. Tracing of an ongoing consultation can be done graphically using bar charts to display the current stage of the ranking, or tracing information can be displayed as text. The amount of tracing can be controlled by the expert.

Dynamic bar charts show intermediate consultation results. The effects of each trait can be seen on the solution set. After watching the effect of a particular trait, the expert may manually adjust the repertory grid information to decrease or increase the effect of the trait on the solution set.

Final consultation results may be displayed as an ordered bar chart. This graphic representation helps the expert see the *intervals* between solutions, while textual lists help the expert see the *overall ordering*. Consultation results can also be displayed in normalized form in a pie chart. It is often easier to see the *relative weight* of each solution in a pie chart (as opposed to the *absolute weights* emphasized in a bar chart).

Each individual rating's contribution to the solution may be emphasized by highlighting those grid elements which come closest to preference ratings given during a consultation. The expert can quickly see the information that contributed to the particular test case results.

All of these lists, graphs, charts, and grids may be saved from consultation to consultation. The expert can compare previous results to current results and see if changes led to improvements. These displays can also be recreated from saved consultations.

Directed Methods - Elicit Results Expectations, Score Results, Revise

The correlation between the expert's expectation of the solution ranking vs. the outcome of a consultation may be calculated. This correlation is used as a measure of the quality of the knowledge base performance for a case. Solutions that are out of place can be used to elicit new information. The expert may partially or completely specify the expected outcome of a consultation.

Scoring results. Results are scored against expectations by computing a rank correlation measure and checking to see whether or not the result is statistically significant. Correlation scores show the expert how well the knowledge base performed, given the expectations. Correlation scores recorded over time for many test cases can show whether or not the knowledge base is improving as revisions are made.

New Triads Based on Expectations. AQUINAS calculates the solution that is most out of place and asks the expert for a new trait based on where the solution *should* have appeared in the list. When this new trait is included in a subsequent consultation the out-of-place solution should be closer to the expected place in the list (Boose, 1986)

Semi-Automated Methods - Suggesting Changes

Solution results adjustment. Implementation of these methods is in progress. If the expert disagrees with the rank ordering of the solutions, he will be able to modify the bar chart displaying the results of the consultation. His new ordering will be used to find potential changes to ratings or trait weights that would lead to the new ranking. Changes to *repertory grid values* may be suggested that would help push target solutions up or down on the results list. Changes to *trait weights* may be suggested that would help push groups of solutions up or down on the results list. After information is modified the consultation will be replayed and the new and old results will be compared. This method of interaction is similar to a spread-sheet: the expert modifies the answer graphically and the data is used to calculate changes.

Automated Methods - Rating and Trait Weight Revision; Synthesizing New Traits

Preliminary results are presented for methods that examine the knowledge in a repertory grid and automatically generate changes that lead to performance improvements.

Automatic Rating and Weight Adjustment. Values of ratings and trait weights may be varied automatically to produce a new repertory grid. Batch test cases are run, and the results of the changes are measured against the expert's expectations. A new change is selected, and the test cases are re-run and scored.

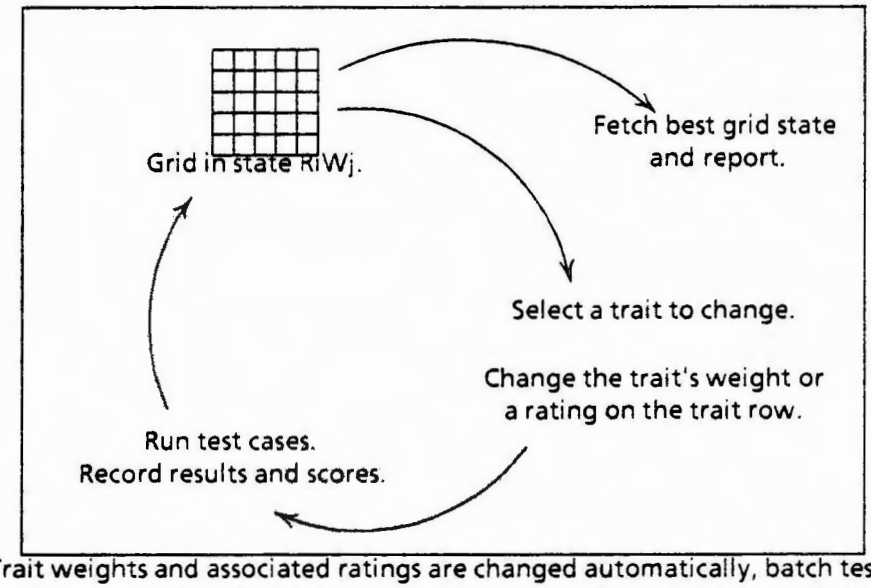


Figure 7. Trait weights and associated ratings are changed automatically, batch test cases are run, and the results are scored against the expert's expectations for those cases and stored. Heuristics are used to select the next trait-weight-rating change. When finished, the best combination of trait weights and ratings is reported to the expert. "Best" is defined as the highest average correlation score over the test case set for that grid state. The expert can examine the proposed grid and direct AQUINAS to implement all or a portion of the suggested changes

Figure 7 shows the grid modification, test, score, and report cycle. The expert starts by specifying a set of test cases and expected outcomes (a rank-ordered solution set for each test case).

Exhaustively checking every possible combination of ratings and weights is not feasible for even medium-sized grids. For example, it took over ninety hours to test a small seven-by-six grid, sampling ratings (1, 3, and 5 on an ordinal scale from 1 to 5), and using six test cases. So, a hill-climbing heuristic is used to select a new change and decide when to stop work. The heuristic is based on:

- the original trait weight (traits with higher weights tend to be tried sooner),
- the original rating distribution spread in the row,
- the distance from the original grid state to the proposed new grid state (closer states are tried first since it is assumed that the grid from the expert is reasonably close to a final state), and
- previous results (if moving a rating or weight in a certain direction produces better results, try pushing it farther in that direction).

Starting with the expert's grid, one change to a rating or weight is made. The change is selected based on the distance of the proposed change from the original grid; "closer" changes are tried before more distant changes. If the change results in overall performance improvement, a change is made from this new state. If the change does not result in an overall improvement, a different change is tried from the old state. If no changes result in performance improvement, a "peak" has been reached. The originally grid is randomly changed periodically to keep from only finding local minima; if the random change produces an overall performance improvement, hill climbing proceeds from there. This process runs until a user-specified time limit is up.

When the process is over, several results are reported:

The grid that exhibited the best performance. The expert can examine the new grid to see whether or not the changes make sense. If the expert agrees, all or some of the changes are automatically implemented in the existing grid.

The "distance" between the best grid and the original grid. Trait ratings and weights in the new grid may be slightly different or significantly different from those in the original grid. When there are significant differences, it may suggest that a new trait should be used that exhibits the new ratings and weights (see the discussion of trait synthesis, below).

A sensitivity ranking for the traits. The overall effect each trait has on all the tested combinations is reported. Traits are ranked according to the greatest performance variance

and reported. This information can help focus the expert's attention on traits that most affect performance.

Local minima. Using the hill-climbing approach described above, grids producing local minima are also reported. These grids may contain optimal interactions between a subset of traits.

A "completeness" score for the test case results. A score is given that shows the overall completeness of possible test case results rankings for the best (or any other) grid (for instance, there would be twenty-five possible outcomes for five possible solutions). This helps the expert see how well the test cases cover the set of solutions. This information can be used to generate new potential test cases (see below).

This technique has produced promising results. Minor sets of changes have yielded significant problem-solving improvement over small sets of test cases.

Analytic and Synthetic Changes. Using AQUINAS, the best case of knowledge base modification (from a maintenance point of view) is when adjustments made solely to trait values and weights result in improvements of (or at lease, do not degrade) all existing cases. These are referred to as *analytic* changes, since no new traits or solutions are added, and a change can be made by analyzing relationships between existing traits and solutions. Usually this is not the case; a given change may improve some case performance and degrade other case performance. When this happens, knowledge must be added and/or deleted. Adding new knowledge is referred to as a *synthetic* change.

Synthesizing New Traits. Examination of the consultation results makes it possible to hypothesize new traits that would continue to keep the knowledge base correct for existing cases. A new trait is formed internally for a set of solutions; its values are systematically varied and combined with existing traits and tested against existing consultations. If the inclusion of this trait improves the overall scores of existing consultations, the expert is asked to try and name it, and it is included in the knowledge base. Such traits can also be used to start a *new* grid (Shema and Boose, 1988). A trait's original ratings could also be changed so much that the expert feels that the meaning of the trait has changed. Again, the expert would try to name the new trait. Trait change measurements are reported to the expert along with the best grid (see above).

Implementation of the following two methods is in progress.

Consultation synthesis. Examining the contents of previous consultations may suggest new test cases that are significantly different from those already present (that is, their rank orders are significantly different from any existing test cases). The expert could evaluate these synthesized test cases and try to see if they fit any real-world situations. The knowledge base validation effort could be assisted by the expanded coverage of these test cases.

Test Case Analysis. Comparisons of consultation results might highlight discrepancies in the expected results that the expert has given. Test cases may not be appropriate for the section of the knowledge base under consideration, or the expert may have made an error in his ranking. Validation of the test cases often is as important as the validation of the knowledge base.

OTHER AQUINAS FEATURES

Strategic and Procedural Knowledge

We have performed experiments with AQUINAS to elicit high level strategic knowledge. We took cases and listed them on the top of the grid, and then asked the expert, "What strategy might you use to solve two of CASE1, CASE2, and CASE3, but not the third?" Strategies were elicited as traits, and then the expert rated the applicability of each strategy to each problem case. This method tended to highlight the priorities of various procedures in solving the problem (Figures 9 and 10).

What strategy might you use to solve two of AUTO.ACCIDENT, MEDICAL.CALL, and MOUNTAIN RESCUE but not the third?
NEW STRATEGY** **FIND OUT LOCATION**

What is that strategy's opposite as it applies in this case?
NEW STRATEGY** **LOCATION NOT ESSENTIAL**

What is the name of a scale or concept that describes FIND.OUT.LOCATION / LOCATION.NOT.ESSENTIAL?
NEW STRATEGY (CONCEPT)** **LOCATION**

```
What strategy might you use to solve two of MEDICAL.CALL, MOUNTAIN.RESCUE, and MISSING.CHILD, but
not the third?
NEW STRATEGY** FIND OUT SEVERITY

What is that strategy's opposite as it applies in this case?
NEW STRATEGY** SEVERITY NOT ESSENTIAL

What is the name of a scale or concept that describes FIND.OUT.SEVERITY / SEVERITY.NOT.ESSENTIAL?
NEW STRATEGY (CONCEPT)** SEVERITY
        :
```

Figure 9. Triadic comparison of cases is used to elicit strategies and procedures for a hot-line dispatch portion of the medical aid advisor.

```
                OBS.SOLUTION.TRAIT
4 5 1 3 1 4   1. (5) LOCATION: FIND.OUT.LOCATION(1)/LOCATION.NOT.ESSENTIAL(5) [ORDINAL
5 1 2 5 5 3   2. (5) SEVERITY: FIND.OUT.SEVERITY(1)/SEVERITY.NOT.ESSENTIAL(5) [ORDINAL]
4 5 2 4 1 2   3. (5) ACCESSIBILITY: FIND.OUT.ACCESSIBILITY(1)/ACCESSIBILITY.NOT.IMPORTANT
2 5 4 5 1 4   4. (5) NUMBER.OF.VICTIMS. FIND.OUT.NUMBER.OF.VICTIMS(1)/NUMBER.OF.VICTIM
1 4 3 5 5 2   5. (5) AMOUNT.OF.EQUIPMENT: FIND.OUT.AMOUNT.OF.EQUIPMENT.NEEDED(1)/AMI
 | | | | | |
 | | | | | 6. LOGGING.ACCIDENT
 | | | | 5. PLANE.CRASH
 | | | 4. MISSING.CHILD
 | | 3. MOUNTAIN.RESCUE
 | 2. MEDICAL.CALL
 1. AUTO.ACCIDENT
OBS.SOLUTION
```

Figure 10. Repertory grid of cases (columns) and strategies (rows) for a hot-line dispatch portion of the medical aid advisor. For this problem, strategies are *procedural priorities* for determining different types of information.

When this information is inverted (rows and columns are exchanged so that strategies become "solutions" and cases become "traits") the type of case can be used to select the appropriate strategy during consultations.

AQUINAS and Machine Learning

Machine learning in AQUINAS takes place in two forms: *interactive* and *automatic*. Interactive forms include implication generation, analysis, and review. Automatic forms include strategies embedded in the inference engine and methods to automatically improve knowledge bases. Analysis and refinement tools in AQUINAS help the expert produce a grid that performs well. Learning mechanisms that start with such knowledge can be very effective. Combining more traditional machine learning techniques with interactive elicitation methods in the same tool provides useful feedback to the user that triggers new insights and that can shed new light on machine learning research.

Interactive Learning. Inductive implications between trait values are computed with an algorithm developed by Gaines based on fuzzy logic (Gaines, 1985). A repertory grid is used as a set of examples or learning set.Trait values are viewed as logical predicates, solutions are the operands of the predicates, and ratings are fuzzy truth values. Implications are shown graphically (as in Figure 2, above) or listed textually. The strength of each implication is shown, either by listing the score or by varying the thickness of the arrow on the graph. Implications show the expert relationships at a higher levels of abstraction implied by a repertory grid. If the expert disagrees with an implication, AQUINAS helps the expert refine the grid. Frequently, the expert can think of an exception to the implication (a new solution) that disproves it. This solution is entered, rated, and the implication strength is reduced appropriately. Sometimes implications point out inconsistencies in the way the expert is using a trait. In such cases laddering is used to help the expert decompose inconsistent traits into consistent sub-traits.

Automatic Learning - Reasoning. In a complex knowledge base the expert may not want to rate every possible cell in every possible repertory grid implied by the four hierarchies (every set of four nodes - one from each hierarchy - defines a unique grid cell). Experts tend to rate the leaf cells of hierarchies leaving cells at higher levels unrated. If the hierarchies are deep the expert may rate no more than ten or twenty per-cent of the existing grid cells. However, AQUINAS' inference engine expects all cells to be rated so that solutions may be scored properly for each observation, preference, or constraint. AQUINAS has several mechanisms for "filling in" missing ratings as needed. Lower level ratings can be abstracted to higher levels (induction); ratings of parent values can be inherited down to child ratings (deduction); best guesses can be made by looking at siblings' ratings if they exist

or by examining the functional similarity of traits (analogy); users can supply their own application-dependent derivation functions. These mechanisms are used directly by the inference engine and to produce derived grids that show AQUINAS' best inferences for missing ratings.

Automatic Learning - Consultations and ID3. An ID3-like algorithm (Quinlin, 1987) may be optionally employed by the reasoning engine during consultations. The mechanism dynamically optimizes question ordering based on ratings in the grid.For example, the user can instruct the inference engine to "only ask the three most important questions at this hierarchical level" or to only ask questions that score above a certain threshold of utility.

Automatic Learning - Automatic Grid Improvement and The New Term Problem. In general, learning systems cannot extend or modify their initial vocabulary to generate new descriptors (new terms) when needed. There are two types of new terms: terms resulting from the compilation or decompilation of existing terms and truly new terms that are orthogonal to the original ones. AQUINAS' clustering mechanism can help experts identify compilations of existing terms when experts are asked to label cluster junstions. AQUINAS' automatic grid improvement mechanism addresses part of the second problem. New unlabeled traits (terms) are produced, and the expert is asked to name them (see discussion above). In this case, AQUINAS identifies the *need for* and *characteristics of* a new term that is guaranteed to improve the performance of the knowledge base and the expert is asked to supply the name of the term. For both types of new terms,allowing the expert to interact with clustering and automatic improvement mechanisms seems to be the key to effectiveness.

Automatically Generating Production Rules from Grids

AQUINAS can transform repertory grids into sets of production rules. These rules can be automatically reformatted for several expert system building tools (KEE, KS-300/EMYCIN, LOOPS, M.1, OPS5, S.1, and others). Certainty factors for rules are generated based on EMYCIN's certainty factor calculus (Adams, 1985). Certainty factor values are based on the relative strength of the rating, the relative weight of the trait, and the overall size of the grid.

Experts find it easier to analyze and maintain knowledge in the more compact form of repertory grids rather than in the form of production rules. In ETS, use of production rules and an internal reasoning engine were the sole methods of reasoning. This method has been replaced in AQUINAS with a maximum entropy-based approach. Rule-based reasoning in AQUINAS can still be used to pre-test knowledge bases destined for other expert system building tools. Certainty factor generation has been fine-tuned so that results using maximum entropy and certainty factors are very close for most applications.

Process Guidance

A subsystem of AQUINAS called the dialog manager contains pragmatic heuristics to guide the expert through knowledge acquisition using AQUINAS. Its help is important in the use of AQUINAS, given the complexity of the AQUINAS environment and the many elicitation and analysis methods available to the expert. The dialog manager makes decisions about general classes of actions and then recommends one or more specific actions providing comments and explanation if desired. This knowledge is contained in rules within the dialog manager in AQUINAS. A session history is recorded so that temporal reasoning and learning may be performed (Kitto and Boose, 1987a, 1987b, 1988). The dialog manager helps novices learn to use AQUINAS, resulting in more efficient prototyping.

Steps in Building a Rapid Prototype with AQUINAS

A typical sequence of steps when using AQUINAS to build knowledge bases is:

1. Elicit cases and the initial grid (solutions, traits, and ratings)
2. Analyze and expand the initial, single grid
3. Test the knowledge in the single grid by running consultations
4. Build hierarchies (structured as solutions and traits in multiple grids) from the first grid if problem is complex
5. Use several rating value types (transform ordinal ratings to nominal and interval ratings) to represent knowledge
6. Test knowledge in hierarchies; add, analyze and test knowledge from other experts if warranted by application
7. If multiple experts are used, analyze the similarities and differences among experts; conduct structured negotiation among the experts (Boose, 1986b)

8. Edit, analyze, and refine the knowledge base, building new cases
9. Further expand and refine the knowledge base by successively testing the knowledge to see if the results agree with the expert's opinion

AQUINAS APPLICATION AREAS

AQUINAS is useful for *analysis* problems whose solutions may be comfortably enumerated, such as classification, interpretation, and diagnosis. It is not as useful for *synthesis* problems where unique solutions are built up from component parts. Such problems include configuration, planning, scheduling, and design. However, AQUINAS can often be used on the analytic components of larger synthesis problems.

Dimensions of AQUINAS Use

AQUINAS can be used in a number of ways to help facilitate decision-making. Using AQUINAS to help build a large, complex expert system is one such use. However, AQUINAS can also be used as a stand-alone one-shot personal decision aid, a teaching aid, a group data gathering and negotiation tool, a tool for exploring project feasibility, and so on.

AQUINAS is useful in a variety of situations because of the flexibility and ease of use of repertory grids and because of its ability to rapidly gather knowledge and build a prototype consultation system.

Dimensions of use of AQUINAS as a knowledge-based decision making aid are listed below.

Source and user: The source of the knowledge may be the same or different than the user of the knowledge.
Source quantity: One person or more than one person may contribute to the knowledge base.
User quantity: One person or more than one person may use the knowledge base.
Knowledge base longevity. Knowledge bases may be used once and "thrown away" after some problem insight has been achieved or they may be used repeatedly, at different rates (continuously, hourly, daily, weekly, monthly, and so on).
Knowledge base update frequency. Knowledge may be updated with varying rates of frequency. For example, a "frozen" knowledge base may be used for a long time before it is updated or a "living" knowledge base may be updated continuously (a large, commercial, time-share knowledge base with few updates, a marketing trends knowledge base with periodic updates, a stock market knowledge base with continuous updates).
Stand-alone: The tool can be used for stand-alone decision use or used with other software systems (expert system shell, spreadsheet; database management system).
Commercial knowledge base use: The knowledge base can be generated for commercial, external or internal, or personal use
Consulting Use: The decision maker can use the tool directly or a consultant can use the tool in support of the decision maker (decision consulting; education and training).
Empty tool or primed tool: The tool can be used as an empty general decision aid or primed with knowledge for specific applications. Knowledge templates or full or almost-full knowledge bases may be delivered to a customer. Some examples are knowledge for specific domains at varying levels of abstraction (general diagnosis, engine diagnosis, jet engine diagnosis), proprietary or commercially valuable knowledge, surveys (fill in the ratings and return the grid to the surveyors), or use of domain-tailored dialogs (such as the original use of repertory grids in psychotherapy where patients filled in specific roles based on supplied descriptions).
Process and product: These tools can be used in the process of making a knowledge base as well as actually delivering knowledge (feasibility analysis, education and training of experts in the knowledge engineering process).

ETS (based on single grids) and AQUINAS (handling multiple hierarchical grids) have been used to generate hundreds of small and medium sized knowledge-based systems in the following categories (Boose, 1988). The categories are derived using different combinations of values for the above dimensions.

1. **Decision aid for for one-shot decisions**
   A grid tool is used to help people gain insight while making decisions: What employee should I assign to this project? What stocks should I invest in? What car should I buy? How can I better represent my products to my customers?

2. **Stand-alone knowledge-based system development and delivery tool**
   A grid tool is used to help develop, deliver, and maintain a knowledge-based system in a cost-efficient manner.
3. **Group decision aid**
   A grid tool is used as an aid to facilitate rapid, documented group decisions: On what software and hardware environment should this project be implemented? What research should our company pursue? How should capital assets be assigned?
4. **Large-scale on-line knowledge bases for a community of users**
   A grid tool can be used to develop and maintain large dial-in knowledge-and-data bases for a large number of users (stock market advising, insurance claims information).
5. **Feasibility analysis and project exploration tool**
   A grid tool is used to help assess project feasibility. What project idea is technically feasible? What project ideas are possible in this general area? Given a project idea, what kinds of knowledge will be involved? Does it make sense to invest resources in this project?
6. **Expert system building tool (shell) front-end**
   A grid tool can be used as a front-end to develop knowledge for another expert system shell (S.1, M.1, OPS5, etc.).
7. **Teaching aid**
   A grid tool can be used to help teach others (especially experts and beginning knowledge engineers) quickly and painlessly about the concepts involved in knowledge-based systems and knowledge engineering: What are production rules? What is a consultation system? How can an expert's knowledge be structured and modified? How can systems be tested and validated?
8. **Situation Insight**
   A grid tool can be used to discover important or controlling factors in a situation; these factors can help the user understand, control, or change the situation (job satisfaction and motivation self-analysis, psychotherapy, counseling, relations with colleagues).

Large knowledge bases (containing thousands of judgments) are typically generated as the result of group decision applications. A twenty-by-twenty grid contains 400 ratings, when such grids are collected from multiple experts the resulting knowledge base contains many thousands of ratings. For example, a past AI tool advisor system included over four thousand ratings from four experts. Larger systems from single users tend to have more hierarchical structure and fewer ratings. For purposes of comparison, a rating can be thought of as the rough equivalent of a context-dependent rule in a rule-based system

DISCUSSION AND FUTURE WORK

AQUINAS inherits the advantages of ETS: rapid prototyping and feasibility analysis, vocabulary, solution and trait elicitation, interactive testing and refinement during knowledge acquisition, implication discovery, conflict point identification, expert system shell production, and generation of expert enthusiasm (Boose, 1986a). It is much easier for users to learn knowledge-based system concepts by using AQUINAS than through reading books or attending classes (i.e., rules are automatically generated and used dynamically in consultations; new vocabulary is incrementally introduced).

Alternative approaches may be tested with little expenditure of resources.

The first expert took about two hours to enter, analyze, and refine the knowledge for the heart dysrhythmias case shown here. Working from the first expert's solutions and traits (but not ratings or weights), the second expert took about an hour-and-a-half to enter, analyze, and refine the knowledge for this case. A knowledge engineer took less than two hours to draw and enter the graphic displays.

Knowledge from multiple experts may be rapidly combined using AQUINAS. Users may receive dissenting as well as consensus opinions from groups of experts, thus getting a full range of possible solutions. Disagreement between the consensus and the dissenting opinion can be measured to derive a degree of conflict for a particular consultation. The system can be used for cost-effective group data gathering (Boose, 1986b).

Elicitation, structuring, analysis, and testing of knowledge is based on specific cases. When knowledge in AQUINAS is updated, it is done so with respect to a specific case. Addition of new knowledge in this way can be strictly controlled by the expert; the tendency for local changes to degrade other cases is thus curbed.

Future Directions

In the future we plan to extend AQUINAS in several areas:

Capturing Explanations. Templates will be used to capture comments from experts when making judgments (ratings, weights, traits from triads). Comments will be examined to help maintain the knowledge base and will be played back during consultations to provide explanations for the end-user. Applicable work on classifying explanations has been reported by Schank (1986) and Kass and Leake (1987).

Further Work in Acquiring and Analyzing Knowledge from Multiple Experts. In addition to AQUINAS' current capabilities to analyze and use knowledge from multiple experts, we plan to implement the following features:

- Elicit knowledge in parallel (on-line) from several experts (Chang, 1986) in the same or different fields.
- Analyze subsumption and overlap of experts' knowledge in multiple fields (similar to SocioGrids in KSS0, Shaw and Gaines, 1987).
- Support on-line structured negotiation among experts to solve problems or make recommendations.
- Discover the most important aspects of individual knowledge when compared to group knowledge (special cases).
- Elicit and merge knowledge dynamically for use in on-line applications.

Knowledge Acquisition for Synthesis Problems. Attempts are being made to use grid tools to elicit knowledge for synthesis problems. For example, we are experimenting with allowing the user to put arbitrary computations in the cells of repertory grids in AQUINAS (spreadsheet-like computations, database access calls, functions that sample sensors). However, this may make the tool more difficult to use for these applications.

Eventually, we hope to blend together techniques from other tools (such as MDIS, Antonelli, 1983, and SALT, Marcus, 1985, 1987) to help elicit knowledge for aspects of design problems:

- Acquisition of causal models and design constraints.
- Selecting alternative design concepts based on competing criteria such as reliability, maintainability, cost, manufacturability.
- Acquiring human expertise in validation and verification of designs.
- Acquiring knowledge for design evaluation and change recommendation.
- Acquiring knowledge for document generation and evaluation.
- Capturing historical data in a "corporate memory" database to help solve future design problems.

Other knowledge acquisition tools are useful in producing knowledge-based systems more rapidly than manual interviewing methods. Knowledge acquisition tools may be characterized by the problem tasks for which they are designed to gather knowledge and the problem-solving method that uses the knowledge to solve a problem (Boose, 1988b). Particular knowledge acquisition tools can be viewed as linking tasks and methods.

AQUINAS attempts to capture knowledge for most types of analysis problems; other systems such as MOLE and MDIS derive their power from knowing more about the application domain.

We intend to build a knowledge acquisition environment that includes specific domain knowledge for specialized application areas and can acquire knowledge for synthetic problems, combining features from other knowledge acquisition tools. Development of the AQUINAS workbench will continue in an incremental fashion. Techniques will be continuously integrated and refined to build an increasingly more effective rapid knowledge acquisition environment.

ACKNOWLEDGMENTS

Thanks to Miroslav Benda, Kathleen Bradshaw, Brian Gaines, Cathy Kitto, Ted Kitzmiller, Art Nagai, Doug Schuler, Mildred Shaw, Suzanne Shema, and Lisle Tinglof-Boose, for their contributions and support. AQUINAS was developed at the Knowledge Systems Laboratory, Advanced Technology Center, Boeing Computer Services in Seattle, Washington.

REFERENCES

Adams, J., Probabilistic Reasoning and Certainty Factors, in B. Buchanan and E. Shortliffe (eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley, 1985

Antonelli, D., The Application of Artificial Intelligence to a Maintenance and Diagnostic Information System (MDIS), in the *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance*, Boulder, CO., 1983

Boose, J. H., Personal Construct Theory and the Transfer of Human Expertise, in the *Proceedings of the National Conference on Artificial Intelligence*, Austin, Texas, 1984.

Boose, J H., A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, *International Journal of Man-Machine Studies*, 23, 1985.

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

Boose, J. H., Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain, *Future Computing Systems Journal*, Vol 1., No. 2, 1986b.

Boose, J. H., Uses of Repertory Grid-Centered Knowledge Acquisition Tools for Knowledge-Based Systems, Special Issue on the Second Knowledge Acquisition for Knowledge-Based Systems Workshop, *International Journal of Man-Machine Studies*, in press, 1988.

Boose, J. H., and Bradshaw, J. M., Expertise Transfer and Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, 1987.

Buchanan, B., and Shortliffe, E. (eds.), Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Reading, MA: Addison-Wesley, 1985.

Chang, E., Participant Systems, *Future Computing Systems*, Vol. 2, No.1, 1986.

Clancey, W., Heuristic Classification, in J. Kowalik (ed.), Knowledge-Based Problem-Solving, New York: Prentice-Hall, 1986.

Davis, R., and Lenat, D. B., Knowledge-Based Systems in Artificial Intelligence, New York: McGraw-Hill, 1982.

Gaines, B. R., and Shaw, M. L. G., Induction of Inference Rules for Expert Systems, *Fuzzy Sets and Systems*, Vol. 8, No. 2, p. 315-328

Gaines, B. R., and Shaw, M. L. G., New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in M L. G. Shaw (ed.), Recent Advances in Personal Construct Technology, New York: Academic Press, 1981

Kass, A., and Leake, D., Types of Explanations, Yale Computer Science Report YALEU/CSD/RR #523, March, 1987.

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

Kitto, C. M., and Boose, J. H., Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No.2, 1987a

Kitto, C. M., and Boose, J. H., Choosing Knowledge Acquisition Strategies for Application Tasks, the Proceedings of Westex-87, Anaheim, 1987b.

Kitto, C. M., and Boose, J. H. (1988). Selecting Knowledge Acquisition Tools and Strategies Based on Application Characteristics, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Marcus, S., McDermott, J., and Wang, T., Knowledge Acquisition for Constructive Systems, in the *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, 1985.

Marcus, S., Taking Backtracking with a Grain of SALT, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*, Vol. 26, No.4, 1987.

Quinlin, J. R., Simplifying Decision Trees, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol. 27, No.3, 1987, p. 221-234.

Saaty, T. L., The Analytic Hierarchy Process, New York: McGraw-Hill, 1980.

Schank, R. C., Explanation Patterns: Understanding Mechanically and Creatively, Lawrence Erlbaum Associates, Englewood Cliffs, NJ, 1986.

Shaw, M. L. G., and Gaines, B. R., Techniques for Knowledge Acquisition and Transfer, special issue on the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 4, *International Journal of Man-Machine Studies*, Vol. 27, No.1, 1987.

Shema, D. B., and Boose, J. H., Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press, 1988.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Learning Redesign Knowledge

Evangelos Simoudis
AI Applications Group
Digital Equipment Corporation
290 Donald Lynch Boulevard
Marlboro, MA. 01752, U.S.A

Abstract

The paper presents a learning component which is incorporated into TLTS, a knowledge-based tool that is used for the evaluation and redesign of printed circuit boards that have signal integrity problems. The knowledge of TLTS is organized around a blackboard architecture. The learning component can form new knowledge sources with redesign knowledge. It does this by compiling the knowledge that is contained in redesign plans that the user creates in cooperation with TLTS. The learning component also acquires redesign meta-knowledge for selecting the most appropriate knowledge source to redesign a faulty circuit. It does this by utilizing the problem solving information that is generated by TLTS and cooperating with the user.

INTRODUCTION

In (Simoudis, forthcoming) we describe a knowledge-based system, called TLTS, which implements a model for performing redesign. This model consists of an evaluation and redesign loop. During the evaluation phase the artifact is evaluated against a set of user specified performance constraints. During the redesign phase the structure of the artifact is altered in order to correct constraint violations. The redesigned artifact is re-evaluated and the loop is repeated if the user's constraints are not met.

This model was applied in the domain of redesigning circuit networks which are laid on printed circuit (pc) boards. In the target domain the violations of the user's constraints are called *signal integrity problems*. Circuit networks (hereinafter circuits) are structures which consist of drivers, (logic gate outputs), receivers, (logic gate inputs), and terminators, (resistors, and clamp diode networks). All the elements of a circuit are interconnected via transmission lines, (etches and connectors).

In studying the model we have been able to identify the various kinds of knowledge which need to be specified. The identified kinds were:

- Circuit description knowledge that includes:
  - Knowledge about component hierarchies.
  - Knowledge about parts hierarchies.

- Knowledge about constraints.

- Evaluation knowledge that includes:
  - Analysis knowledge.
  - Diagnosis knowledge.

- Redesign knowledge.

We have also been able to identify the various types of constructs that must be used for the effective representation of the kinds of knowledge described above. For circuit redesign it was determined that we needed to represent:

- *Redesign operators*, whose application changes the structure of a circuit.
- *Operator scheduling rules*, whose task is to resolve conflicts among operators during each redesign step.
- *Eligibility demons*, which are invoked to notify that a collection of operators is capable of redesigning a faulty circuit. [1]
- *Meta-rules*, whose task is to decide which of the eligible collections of operators will redesign a faulty circuit.

This paper describes a learning component which acquires *all* four types of redesign knowledge presented above. It does this by interacting with the performance system, TLTS, and the user. Because of this interaction, the component does not have to separately explain the training examples that are provided by the user in order to determine why each example is an instance of the concept that is to be learned. The first three types of redesign knowledge are extracted from redesign plans that the user forms in cooperation with TLTS. Rather than forming a macro rule that contains the knowledge in a plan and then incorporating the macro rule into the performance system, or storing entire plans and then redesign a faulty by trying to replay a stored plan, the learning component extracts each type of knowledge, generalizes it and expresses it using the appropriate representation. As a result, the acquired knowledge is organized in autonomous bodies which can then be used for the redesign of a variety of faulty circuits rather than circuits identical to the circuit for which the plan was created, as it would had been the case if the plans could only be replayed. Finally, because of the way the acquired knowledge is expressed using the above defined constructs, and organized in autonomous bodies, it is easier to inspect, maintain, and modify whenever the need arises.

Background on TLTS

In order for the reader to better understand the work described in this paper it is essential that a short presentation of TLTS is given. As was mentioned earlier, the operation of TLTS consists of evaluating a circuit and then redesigning it if the evaluation reveals signal integrity problems.

[1] In addition to the three types of redesign knowledge that were mentioned above, a set of low level redesign goals are associated with each collection of operators. The operators are trying to achieve these goals.

Evaluation in TLTS consists of the following phases:

1. *Simulation.* During this phase the behavior of the circuit is obtained.
2. *Analysis.* The analysis phase consists of comparing the circuit's simulation results to the constraints imposed by the user and detecting constraint violations. Each violation is associated with a signal integrity problem.
3. *Diagnosis.* If signal integrity problems are detected in a circuit the diagnostic knowledge determines the causes of each problem.

If signal integrity problems are revealed during the evaluation the circuit is redesigned in order for the detected problems to be eliminated. Since there may be many ways of redesigning a circuit in order to solve a signal integrity problem the most appropriate way(s) has to be first selected. Once this is done a plan of how to redesign the circuit is formed and then applied. The operators in the plan change the structure of the faulty circuit.

The knowledge in TLTS was organized around a blackboard architecture. Each of the simulation, analysis, and diagnosis modules, as well as each of the ways to redesign a faulty circuit is represented as a *knowledge source*. The knowledge for selecting which of a set of competing knowledge sources will operate on the blackboard was included in a special type of knowledge source called the *control knowledge source*. The blackboard contains the circuit that is being evaluated, the constraints placed by the user, the simulation results, the signal integrity problems associated with the circuit, and the causes of each signal integrity problem.

A blackboard architecture was used for the following reasons:

- Supports modularity. A uniform interface between each knowledge source and the blackboard is established. The existence of such an interface allows the independent development of each knowledge source. The various types of knowledge within each knowledge source are represented separately and can thus be developed independently.
- Guarantees expandability. New knowledge sources, particularly redesign knowledge sources, that are learned by the learning component can be easily integrated into TLTS. As long as each knowledge source has the same interface to the blackboard, as all the other knowledge sources, and the learning component guarantees that it does, the addition of the new knowledge source does not influence the operation of the entire system. In a similar way a knowledge source can be taken out of the system without having side effects on TLTS' operation.

Paper Organization

In the next section of the paper we present in detail the structure and operation of TLTS's redesign component. Section 3 describes how the system forms new redesign knowledge sources. In section 4 we present how the system learns redesign meta-knowledge. Related work is presented in section 5. Section 6 discusses our future work and section 7 presents the conclusions of our work to date.

CIRCUIT REDESIGN

In order to redesign a faulty circuit TLTS uses the topology and geometry of the circuit, any user established constraints, the signal integrity problem(s) that were detected, and the cause of each problem. The signal integrity problem(s) and the causes act as high level goals that the redesign knowledge sources have to achieve. As was mentioned above, for each signal integrity problem that can be identified through evaluation there may exist several redesign knowledge sources.

Redesign Knowledge Sources

The eligibility demons are represented as production rules. The Left Hand Side (LHS) of each such rule specifies a set of conditions under which the knowledge source can operate on the blackboard. In the case of the redesign knowledge sources these conditions take into account the analysis, and the diagnosis results. For example, the eligibility rule called *remove-receivers-eligible1*

remove-receivers-eligible1

IF a propagation delay problem has been detected ← analysis result
AND the cause is due to the capacitance of the receivers ← diagnosis result
THEN the knowledge source *remove receivers* is eligible

specifies one set of conditions under which the knowledge source called *remove receivers* can operate on the blackboard.

When a redesign knowledge source is invoked, a plan of how to redesign the faulty circuit has to be formed out of the operators that belong to the action part of the knowledge source. The first step in this process is for all the goals that the knowledge source contains to be posted on the blackboard. This does not imply that all of the goals need to be achieved in every plan which is formed out of the operators. It does imply however that at least some of the goals will need to be achieved. For example, removing a receiver from a circuit involves two steps:

1. Removing the receiver,
2. Removing the circuit element that connects the receiver to the rest of the circuit. Such an element may be a connector, a piece of etch, or a via.

The knowledge source called *remove receivers* mentioned in the eligibility rule whose definition was given above has the following goals: remove–a–receiver, remove–a–piece–of–etch, remove–a–via, remove–a–connector. Assume that receivers *re8* and *re4* that belong to the circuit shown in figure 1 are responsible for the propagation problem that was detected. In a plan to eliminate these receivers from the circuit the goal remove–a–piece–of–etch will be achievable, whereas the goals remove–a–via, and remove–a–connector cannot be achieved.

A plan consists of instantiations of the redesign operators which belong to the knowledge source that has been invoked. Each redesign operator is also represented as a production rule. The LHS of each operator describes the conditions under which a redesign operation may be performed. These conditions express circuit topology characteristics and constraints placed by the user/designer on

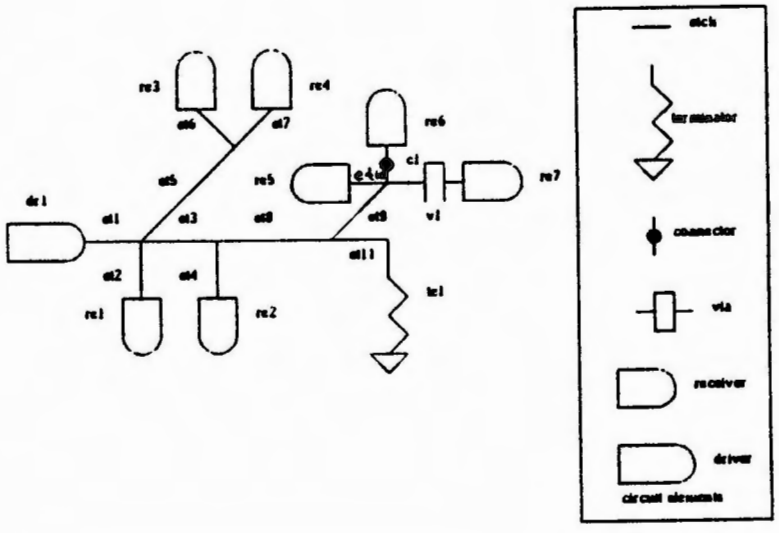


Figure 1: **Circuit to be redesigned**

the circuit element on which the operator is applied. Every redesign operator can belong to several knowledge sources. For example, the operator called *remove-etch* is part of the knowledge sources called *remove receivers* and *shorten etch lengths*.

With each operator we also associate the redesign goal that the operator can achieve. When the conditions in the LHS and the goal the operator can satisfy match against the contents of the blackboard the operator is considered for inclusion in the redesign plan being formed.

The Right Hand Side (RHS) of a redesign operator describes the operation, and takes care of the side effects that result from the operation. For example, if a receiver is removed from the circuit then all the circuit elements that were connected to it have to be informed of this change. This is done by sending messages to the object instances which represent these circuit elements. The operator, called *remove-etch*, is shown below. A piece of etch is removed whenever only one of its two ends is connected to another circuit element.

remove-etch

IF no network element is connected to the one end of an etch
AND the user has specified that the piece of etch can be removed
THEN remove this etch from the network
GOAL remove–a–piece–of–etch

During the plan formation process, depending on the topology of the faulty circuit, some operators may be used in the plan more than once, whereas others may not be used at all. For example, the knowledge source called *remove receivers* has the operators with the following names: *remove-receiver*, *remove-etch*, *remove-connector*, *remove-via*, *move-terminator*, *retrieve-device*, *place-device*,

place-etch, and *place-connector*. If receivers *re3*, and *re4* need to be removed from the network in figure 1 the following actions are involved: (1) removing the receivers *re3*, and *re4* (performed by the operator *remove-receiver* which will need to be included in the plan twice) (2) removing the etches *et5*, *et6*, and *et7* (performed by the operator *remove-etch* which will need to be included in the plan three times).

Determining the sequence in which the redesign operators will be executed in a plan is very important. Inappropriate sequencing will lead to interactions which may prevent a redesign plan from succeeding. For this reason it is necessary to associate operator control knowledge with each redesign knowledge source. The control knowledge is in the form of *scheduling rules*. The rule for resolving the conflict between the operators *remove-receiver* and *remove-etch* that was encountered in step (2) of the example given above is the following:

resolve-operator-conflict1
IF the operator *remove-etch* is eligible
AND the operator *remove-receiver* is eligible
THEN prefer the operator *remove-receiver*

The scheduling rules that are associated with a set of redesign operators are invoked at each step of the redesign plan formation process. These rules accept all the redesign operators that are eligible to be applied during a redesign step and decide which operators should be applied in the step. They also decide how many instantiations of a particular operator will be executed during a redesign step. For example, the operator called *remove-receiver* is instantiated twice in the course of redesigning the circuit in figure 1. Both instantiations can be applied during the same step or they can be applied during separate steps.

Once the operators in a plan and their sequence are determined, the plan is applied on the circuit being redesigned. The plan for removing receivers *re3* and *re4* is the following: (1) remove receiver *re3*, (2) remove receiver *re4*, (3) remove etch *et6*, (4) remove etch *et7*, and (5) remove etch *et5*.

The execution of a plan does not guarantee that the signal integrity problem is eliminated from the faulty circuit. Evaluation follows the redesign phase. If the evaluation reveals that signal integrity problems are still present, the diagnosis and redesign phases are repeated. The capability of TLTS to interleave evaluation and redesign allows the tool to determine how to proceed during the next pass through the evaluation–redesign loop.

Control Knowledge Source

Selecting which of the eligible knowledge sources will operate on the blackboard is particularly important in the case of the redesign knowledge sources. An inappropriate or random choice of redesign knowledge source may delay or make impossible the attainment of a problem–free circuit. It may also lead to signal integrity problem interactions. Namely, while trying to solve one signal integrity problem others may be created or future modifications of the circuit may be impossible because of the redesign knowledge source chosen. This form of control knowledge is expressed in *meta–rules*. The meta–rules are placed in a special type of knowledge source called *control*

knowledge source.

Each meta–rule accepts the analysis and diagnosis results, possibly the secondary user constraints, and a set of eligible knowledge sources and determines which knowledge source will operate on the blackboard. An example meta–rule is the following:

resolve-ks-conflict1

IF there exists a propagation delay problem
AND the problem is due to the capacitance of the receivers
AND space exists on the pc board
AND the knowledge source *remove receivers* is eligible
AND the knowledge source *shorten etch lengths* is eligible
THEN select the knowledge source *remove receivers*

This selection is made because the knowledge in the knowledge source *remove receivers* removes receivers from a circuit and forms a different circuit with them. Since there is space on the board this new circuit will be able to be laid out. Hence the selection of the knowledge source *remove receivers.*

FORMING NEW REDESIGN KNOWLEDGE SOURCES

In order to form a new redesign knowledge source the learning component needs to acquire the low level redesign goals that the operators in the knowledge source will be able to achieve, the redesign operators which will achieve these goals, the knowledge for controlling the sequencing of these operators, and the eligibility demons. These types of knowledge are acquired by having the learning component observe the problem solving activity which is followed by TLTS in cooperation with the user in various situations (contexts). The learning component then acquires the appropriate knowledge by examining the problem solving information which was created, and compiling redesign knowledge from it. In particular, the learning component follows a four step process in order to form a new redesign knowledge source. The steps are the following:

1. The learning component record the constraints placed on the circuit by the user, and the evaluation results (analysis and diagnosis) generated by the corresponding knowledge sources.
2. The user cooperates with TLTS and dynamically creates plans of how to redesign a faulty circuit.
3. The plans are saved by the learning component.
4. After a number of plans have been captured the user issues a command and the learning component extracts:
   (a) the redesign knowledge (goals, operators and scheduling rules) these plans contain and forms a new knowledge source,
   (b) the eligibility demons.

Throughout the rest of the paper, and in order to demonstrate how a new redesign knowledge source is formed, we will use examples appropriate for the definition of the redesign knowledge

source called *remove branches*.

Recording Evaluation Results

In addition to the given circuit and the user constraints, the learning component records:

1. The analysis knowledge that is used for the identification of the signal integrity problem.
2. The diagnosis knowledge used for the detection of the cause of a signal integrity problem, if such a problem exists.

For the purposes of an example assume that the evaluation of the circuit shown in figure 1 has revealed two signal integrity problems: a signal strength problem, and a propagation delay problem.

The rule called *detect-signal-strength1*, whose definition is given below, reveals the signal strength problem.

detect-signal-strength1

IF the current that circuit's driver can source is greater than 0
AND the difference between the current the driver can source and
the current the receivers can sink is less than 0
THEN deduce signal strength problem

The rule called *detect-prop-delays1*, whose definition is given below reveals the propagation delay problem.

detect-prop-delays1

IF the first receiver in a circuit violates the timing constraints imposed by the user
AND at least one more receiver violates timing contraints imposed by the user
THEN deduce propagation delay problem

The diagnostic rule called *diagnose-prop-delay1* identifies the cause of the propagation delay problem as being the capacitance of a cluster of receivers in the circuit.

diagnose-prop-delay1

IF propagation delay problem has been deduced
AND the circuit has a cluster of receivers
THEN the cause of the problem is the capacitance of the cluster

The learning component keeps track of the instantiations of the analysis and diagnosis rules that are created. Through a backchaining process it determines the inferencing sequence that links the instances of the analysis and the diagnosis rules that identified each signal integrity problem and its corresponding cause. One such sequence links instantiations of rules *detect-signal-strength1* and *diagnose-signal-strength1*. Another links the instantiations of the rules *detect-prop-delay1* and *diagnose-prop-delay1*. The user selects which problem he will try to solve through redesign. The rule sequence that has identified this problem is marked so that the redesign plan which will be

later created is associated with it.

Creating Redesign Plans

In order to create a plan of how to redesign the faulty circuit the user selects goals from a catalog of predetermined goals and posts them on the blackboard. TLTS also maintains a catalog of redesign operators that can achieve these goals. As soon as the goals are posted on the blackboard all the redesign operators examine the blackboard. The operators that can satisfy the posted goals become eligible for execution. The user instructs TLTS to apply one of the eligible operators. After the operator is executed the above cycle repeats, namely the user chooses another operator etc., until all of the user's goals have been achieved.

In this mode of operation we are making two fundamental assumptions:

1. The redesign operators used in the plans must already exist in TLTS' catalog. This is not an unreasonable assumption in the domain of pc board redesign because the set of redesign transformations that can be performed on pc boards is small and does not change as new pc board technologies are created. In other domains such as VLSI circuit redesign the number of possible transformations is large enough, and constantly changing as new technologies are being developed, thus making hard the creation of a catalog, which can be reused over time. of all the possible transformations.
2. If a redesign operator is missing the user can define it and incorporate it into the catalog. This may not be a realistic assumption because it expects the user to learn the syntax of the operators and define the operation in terms of the programming language used in the system.

The plan formation process is illustrated in the following example. Suppose TLTS is trying to solve the propagation delay problem that has been revealed on the circuit of figure 1. Assume that the user would like to create a redesign knowledge source which he calls *eliminate branches*.

Suppose the goal catalog has the following goals: 1) remove–a–driver, 2) remove–a–receiver, 3) remove–a–piece–of–etch, 4) remove–a–connector, 5) remove–a–via, and that the user would like to remove receivers *re5* and *re6* from the circuit.[2] The user first marks the two receivers which will be removed and then selects the goals remove–a–receiver, remove–a–connector, and remove–a–piece–of–etch. The operator *remove–receiver* is instantiated. The user instructs TLTS to apply the operator and receiver *re5* is removed. Once this receiver is removed the operator *remove–etch* is instantiated. The operator *remove–receiver* is also instantiated. The user instructs TLTS to apply the operator *remove–receiver* removing receiver *re6*. After this step the operator *remove–connector* is instantiated. So next the user has to choose between the instantiation of the operators *remove–etch* and *remove–connector*. He chooses the operator *remove–etch*, which is then applied, followed by the selection and application of the operator *remove–connector*. At this point the plan has been completed and it has to be stored by the learning component.

Storing Redesign Plans

In order to compile the knowledge of a set of redesign plans into a knowledge source, the context

[2]Notice that receivers *re5*, *re6*, and *re7* in the circuit of figure 1 form a branch.

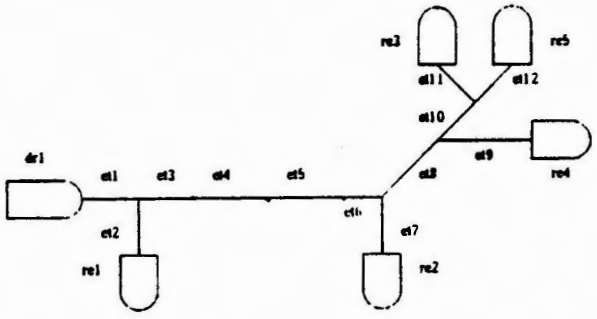


Figure 2: **Second circuit**

of each plan needs to be stored along with the plan itself. The context of a plan consists of the following:

1. The operators the plan contains.
2. The signal integrity problem present in the circuit being redesigned. ← saved during the evaluation phase.
3. The cause of the signal integrity problem. ← saved during the evaluation phase.
4. Any operators which were eligible for execution during each cycle of the plan creation process in addition to the operator which was selected for execution.
5. The structure of the circuit after each redesign operator is applied.

Each plan, the eligible operators that were not used in the plan, and the structure of the redesign circuit are linked with the evaluation results. The plans that are created for the solution of the same signal integrity problem which is due to the same cause are organized into a set. The set is given the name of the knowledge source which will be formed out of the compiled knowledge in the plans. The name is provided by the user. This name is also used by the learning mechanism for accessing the set.

Compiling Redesign Knowledge

The learning component compiles the redesign knowledge which is contained in a set of plans and forms a new knowledge source whose name gave access to the set of plans. In particular, the learning component extracts the goals that were achieved in each plan. It then includes in the knowledge source the union of the extracted goals. As a second step the learning component generalizes the operators which are contained in the plan and includes in the knowledge source the union of the generalized operators. For example, assume there are two plans in the set which is indexed under the name *eliminate branches*. The first plan is the one created for the redesign of the circuit in figure 1. The second plan is used for the removal of receivers *re3* and *re5* from the circuit of figure 2. The plan is the following: (1) remove receiver *re3*, (2) remove receiver *re5*, (3) remove etch *et11*, (4) remove etch *et12*, and (5) remove etch *et10*.

The low level redesign goals which will be included in the new knowledge source are: remove–a–receiver, remove–a–piece–of–etch, and remove–a–connector, because these are the only three goals

that were achieved in the two plans. The operators which will be included in the new knowledge source are: *remove-receiver*, *remove-etch*, and *remove-connector* because the first plan uses the operators *remove-receiver*, *remove-etch* and *remove-connector*, whereas the second plan uses the operators *remove-receiver* and *remove-etch*; their union being the operators *remove-receiver*, *remove-etch*, and *remove-connector*.

Out of the operator precedence relations, as these were established by the sequence of application of each operator in a particular plan, the learning component forms the scheduling rules which are associated with the operators. A scheduling rule which is formed from the operations in the first plan is the following:

sched-rule0

IF the operator *remove-etch* is eligible for execution
AND the operator *remove-connector* is eligible for execution
THEN prefer the operator *remove-etch*

Many of the redesign knowledge sources that TLTS contains include scheduling rules which use domain specific knowledge. In the current implementation of the learning component these types of control knowledge cannot be captured, even though the precedence relations contain implicit domain specific knowledge.

In order to form the eligibility rules (demons) that correspond to the knowledge source which is being formed, the learning component utilizes parts of the captured information about the context of each plan. As it was mentioned earlier in the paper, the learning component stores the inferencing chain of the analysis and diagnosis rule instantiations that detected each signal integrity problem and its cause. The LHS of the rules in the chain that is associated with the problem being solved are used as the LHS of an eligibility rule. Therefore, an eligibility rule is formed out of each plan in the set. If more than one analysis rule and more than one diagnosis rule belong to a particular inferencing chain then the LHS predicates of all the rules are used in the LHS of the formed rule, rather than the predicates of only the first analysis and the first diagnosis rule, because each predicate represents one of three facts: (1) a constraint violation, (2) a cause for a particular violation, (3) a characteristic of the circuit's topology. The RHS of every such rule contains the consequent: (is-eligible *knowledge-source-name*). For example, one eligibility rule which can be formed from the context of the two plans that are being compiled into a knowledge source is the following:

eligibility-rule4

IF the first receiver violates the timing constraints imposed by the user
AND at least one more receiver violates the timing constraints imposed by the user
AND the circuit has a cluster of receivers
THEN the knowledge source *eliminate branches* is eligible

A new redesign knowledge source is developed incrementally. Even after a set of plans are compiled, the user may still have to dynamically create plans because the knowledge source could be missing

redesign operators. For example, assume that the user would like to use the knowledge source that was created out of the two plans above. Only this time he would like to remove receivers *re6* and *re7* from the circuit of figure 1 in order to attempt to solve a propagation delay problem. The reader can see that removing receiver *re7* will also require removing via *v1*. Unfortunately, the new redesign knowledge source does not have an operator for removing vias because such an operator was not used in any of the plans that were utilized while forming the new knowledge source. In this case, the user will have to dynamically create a plan in order to deal with this situation. After capturing this plan the user issues the appropriate command to the learning component so that the knowledge of the plan will be compiled into the knowledge source. The learning component takes the union of the operators that have already been incorporated into the knowledge source with the operators in the plan being compiled and creates and thus updates the set of redesign operators which the knowledge source contains.

LEARNING META-KNOWLEDGE

There are two instances when knowledge needs to be acquired. The first is after a redesign knowledge source is created. At those times the user of the learning component has to instruct TLTS when to select this new knowledge source in relation to the other redesign knowledge sources. The second instance is when a design group has different criteria for choosing among the set of redesign knowledge sources that TLTS already has. In both instances, the same process is used for capturing new meta-knowledge. In particular, given a faulty circuit the learning component:

1. Records the circuit performance specifications provided by the user, secondary constraints placed by the user such as: type of manufacturing process, layout cost, etc.[3], and problem solving (evaluation and redesign) information that is generated by TLTS.
2. Forms meta-rules.
3. Refines meta-rules that have been previously formed.

Recording Information

The learning component starts recording the performance specifications provided by the user as well as secondary constraints only after the circuit is found to be faulty by the analysis and diagnosis knowledge sources. The learning component keeps track of the instantiations of the analysis and diagnosis rules and creates the inferencing chains that link the analysis with the corresponding diagnosis rule instantiations. After this point the redesign knowledge sources examine the blackboard and express eligibility for operating on it, namely redesigning the faulty circuit. While meta-knowledge is being acquired the user always makes the decision of which knowledge source will operate on the blackboard, even when there exists a meta-rule that expresses the user's choice. The learning component records the user's choice. If the choice was not one made by an existing meta-rule or if there are no meta-rules in the knowledge source, the choice is saved for future use. The learning component also records the names of the redesign knowledge sources which are eligible for operating on the faulty circuit.

Once the chosen redesign knowledge source operates on the blackboard and the circuit is redesigned.

[3]Such constraints may need to be taken into account while choosing a way to redesign the faulty circuit.

the learning component records the analysis and diagnosis rules which are instantiated during the evaluation phase of the redesigned circuit. When no analysis rules are executed then it is assumed that the circuit has no signal integrity problems. At this point the learning component utilizes the recorded information in order to form meta–rules.

Forming Meta–Rules

In order to form the LHS of a new meta–rule the learning component uses the results of analysis rules, the results of diagnosis rules, the secondary constraints that were established by the user, and the names of the knowledge sources that were eligible for execution. The RHS of the meta–rule contains the name of the redesign knowledge source that was selected by the user and invoked.

For example, assume that after the evaluation of the circuit shown in figure 1 the user has decided to solve the signal strength problem. In addition the user has determined that there is no available space on the pc board. Further suppose that the following two redesign knowledge sources have expressed eligibility for operating on the blackboard: 1) *change circuit's driver*, and 2) *add another driver* in parallel to the existing one. Finally we will assume that the user chooses the first knowledge source. It has to be mentioned here that the system has no means of determining if the user is making a correct design decision when selecting a redesign knowledge source, at the time the decision is made. Later, we describe a mechanism for avoiding to incorporate inappropriate meta–knowledge. The meta–rule, called *meta–rule1*, that was formed from the example above is:

meta–rule1

IF a signal strength problem is detected
AND the cause of the problem is attributed to the driver
AND no additional space exists on the pc board
AND the knowledge source called *change circuit's driver* is eligible
AND the knowledge source called *add another driver* is eligible
THEN select the knowledge source called *change circuit's driver*

In order for this meta–rule to be later selected and applied by TLTS, the conditions of a new problem situation will have to match exactly those in the LHS of the rule. This is not always possible especially because the secondary criteria vary between designs and design processes. For this reason, the learning component also creates an abstracted version of the above meta–rule. The LHS of every abstracted rule consists of the RHS of the analysis and diagnosis rules that detected the signal integrity problem and its cause. The LHS of the abstracted meta–rule also contains the names of the redesign knowledge sources that were eligible for operating on the blackboard. The RHS of the abstracted meta–rule contains the name of the redesign knowledge source that was selected by the user. The definition of the abstracted rule, called *meta–rule2*, that corresponds to rule *meta–rule1* is:

meta–rule2

IF a signal strength problem is detected
AND the cause of the problem is attributed to the driver
AND the knowledge source called *change circuit's driver* is eligible

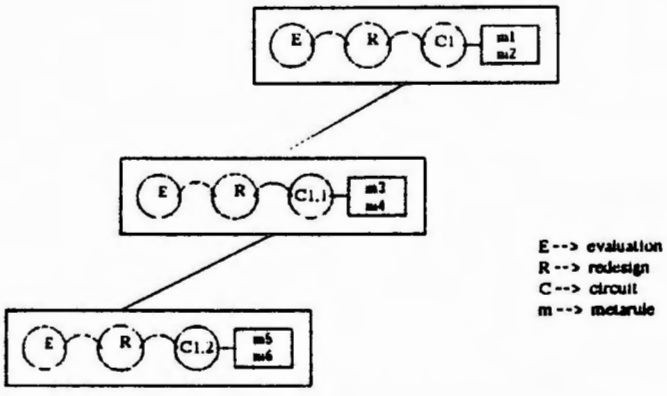


Figure 3: **Backtracking during redesign**

AND the knowledge source called *add another driver* is eligible
THEN select the knowledge source called *change circuit's driver*

The two formed meta–rules are not incorporated into the control knowledge source as soon as they are created. Instead they are attached to the frame which contains the information that was used in creating the pair. Meta–rules are only incorporated into the control knowledge source when the problem solving activity results in a problem free circuit. If the application of the selected redesign knowledge source has not resulted in a problem free circuit the process described above is repeated (namely, the user selects the signal integrity problem which he will try to solve, if more than one is present on the circuit, the learning component creates the inferencing chain of the rules that detected the problem and its cause, the user selects the eligible redesign knowledge source of his choice, etc.) and another pair of meta–rules is created. If the problem is still not solved and the same redesign knowledge sources express eligibility again for operating on the blackboard, the user has to backtrack to some previous version of the circuit and follow a new redesign strategy.[4] In such instances all the meta–rules that have been associated with versions of the circuit that have been created after the circuit to which the user decides to backtrack are eliminated. In figure 3 if the user wants to backtrack to circuit1 then the meta–rules that are associated with circuit1.1 and circuit1.2 are erased. This is the only way that the user can recover from making inappropriate design decisions, and avoid placing incorrect meta–knowledge in the control knowledge source. Once a circuit has been redesigned successfully, the meta–rules that are formed during the problem solving session are stored in the control knowledge source.

Since the learning component creates two versions of each meta–rule it is conceivable that under certain situations both meta–rules will match against the contents of the blackboard and will be eligible to be applied. To avoid this problem a meta–rule control strategy is associated with the rules in the control knowledge source. This strategy selects the most specific of the meta–rules which match against the contents of the blackboard. Therefore, in a situation when both *meta–rule1* and **meta–rule2** match against the contents of the blackboard and are eligible for execution. *meta–rule1* will be selected by the control strategy.

[4]This process corresponds to backtracking in order to search a different part of the problem space.

Refining Meta–rules

It is often the case that once incorporated into the control knowledge source, the meta–rules need to be refined. For example, suppose that after rules *meta–rule1* and *meta–rule2* are learned by the system, TLTS is given to evaluate a circuit, which will be laid on a pc board with extra space, and that the cost of a redesign is not an issue.

Let us further suppose that the analysis and diagnosis knowledge sources of TLTS reveal that there is a signal strength problem which is attributed to the current sourcing capabilities of the circuit's driver. For the same reasons as in the previous example, the same two redesign knowledge sources are eligible for operating on the blackboard. However, only **meta–rule2** will be instantiated by the matcher because its conditions (analysis and diagnosis results) match against the current contents of the blackboard. *Meta–rule1* will not match because some of its conditions (those corresponding to the secondary criteria) do not match against the contents of the blackboard. TLTS next applies *meta–rule2*. As a result it is suggested to the user to select the redesign knowledge source called *change circuit's driver*.

Assume the user knows that since there are no pc board space limitations it is *better* to add another driver to the circuit in order to solve the signal strength problem rather than changing its driver. This means that the suggestion made by the meta–rule *meta–rule2* is not accurate enough. In order to proceed with problem solving the user asks for a list of the other available redesign knowledge sources that were eligible for execution at the time the selection was made by *meta–rule2*. He is then informed that the only other eligible knowledge source is called *add another driver*. Suppose that the user chooses the *add another driver* knowledge source and that after this knowledge source is invoked and applied, the signal strength problem is eliminated. In this case a new meta–rule needs to be created capturing the current situation. Furthermore, a meta–rule with LHS identical to that of *meta–rule2* and RHS expressing the new choice, needs to also be created. .

The first new meta–rule that is created in a way similar to the one previously described above is:

meta–rule3

IF a signal strength problem is detected
AND the cause of the problem is attributed to the driver
AND additional space exists on the pc board
AND no constraints on the cost of redesign exist
AND the knowledge source called *change circuit's driver* is eligible
AND the knowledge source called *add another driver* is eligible
THEN select the knowledge source *add another driver*

The new abstracted meta–rule is:

meta–rule4

IF a signal strength problem is detected
AND the cause of the problem is attributed to the driver
AND the knowledge source called *change circuit's driver* is eligible

AND the knowledge source called *add another driver* is eligible
THEN select the knowledge source *add another driver*

It may be the case that two meta-rules have the same analysis and diagnosis results incorporated in their LHS but different secondary criteria and different RHS. For example, the following two meta–rule are such a pair:

meta–rule1
If a signal strength problem is detected
And the cause of the problem is attributed to the driver
And no additional space exists on the pc board
And the knowledge source called *change circuit's driver* is eligible
And the knowledge source called *add another driver* is eligible
Then select the knowledge source called *change circuit's driver*
meta–rule5
If a signal strength problem is detected
And the cause of the problem is attributed to the driver
And no constraints on the cost of redesign exist
And the knowledge source called *change circuit's driver* is eligible
And the knowledge source called *add another driver* is eligible
Then select the knowledge source called *add another driver*

If the user has established that no space exists on the pc board and that no constraints on the cost of the redesign exist, then both *meta–rule1* and *meta–rule5* will be eligible for execution. Furthermore, since they have the same number of conditions the control strategy, which is associated with the rules in the control knowledge source, will not be able to select the one of the other meta–rule. In such cases both redesign knowledge sources will be executed concurrently by TLTS in a TLTS–unsupervised mode.

RELATED WORK

Two systems use similar techniques to the ones presented in this paper in order to capture design/redesign knowledge. The system Bogart described in (Mostow and Barley, 1987) can acquire new redesign plans. However, Bogart provides no means for grouping related plans. The captured plans can be reused only if they can be replayed. Bogart lacks a component which could adapt portions of or entire existing plans and apply them, by analogy, to different situations. In Bogart either the user chooses which part of a plan to replay or a plan is replayed as much as it can. This, in conjunction with the fact that related plans are not grouped together, implies that the user has to search a potentially large number of plans, understand the context under which they were created, and then decide if he would like to manually retrieve any of them. To the contrary the learning component described in this paper captures separately the goals that are specified by the user, generalizations of the operators used to achieve these goals, and preferences in the execution sequence of the operators. From the captured knowledge, new plans are created every time a circuit needs to be redesigned. As a result, the user does not have to deal with the plan retrieval problem. The fact that the created plans perform local changes in a faulty circuit means that the size of

the plans, and consequently the size of the planning process, does not make replanning prohibitive. Furthermore, since plans are created from scratch, deciding when to replay a portion of a plan, a process which requires analogical reasoning (Carbonell 1983a; Carbonell 1983b) is not needed.

Because the knowledge in the individual plans is not compiled, as the number of captured plans increases Bogart does not become more autonomous. The redesign capabilities of the learning component presented here increase as the knowledge of more plans is compiled into the knowledge source, provided that the plans contain new knowledge. Therefore, the capabilities of the each new redesign knowledge source are dependent to the quality of the examples (plans) that are provided by the user.

In Bogart no other types of knowledge which may play a role in the performance of the system are distinguished except the redesign operators. In particular, Bogart does not acquire any type of control knowledge. As a result, even though other types of knowledge exist in the captured plans, such knowledge cannot be extracted and utilized under Bogart's current knowledge representation.

The system ARGO, which learns both meta-rules and design implementation knowledge is presented in (Acosta, Huhns and Liuh, 1986). However, ARGO is not able to distinguish between the two types of knowledge due to the limited capabilities of the knowledge representation mechanism underlying it. As a result in order to apply a plan under two different conditions two separate rules will have to be formed, with different LHS but identical RHS. This requirement presents a problem to the space that the system needs once a large number of such rules are accumulated. The learning component of TLTS captures only generalizations of operators and not of plans. More specifically, redesign plans are not stored after the knowledge they contain is compiled into a knowledge source. Consequently, the learning component need only acquire the conditions under which an entire knowledge source is applicable and not of each a specific plan.

The micro rules (operators) in ARGO which form a plan are compacted into a macro operator after the plan is formed. However, when one of the micro rules changes as a result of an optimization or correction action, the knowledge representation of ARGO does not allow for the changes to be automatically reflected to all the plans that use the micro operator which has changed. Therefore, all the captured plans need to be examined, and the ones affected by the changes need to be re-implemented. By generalizing the operators that were used in a collection of plans and by compiling them into a redesign knowledge source, changes to the redesign operator level are possible in our system. These changes will then be taken into account the next time the operators of the knowledge source are used to form a plan.

A third system which uses learning methods similar to the ones presented in this paper is SOAR. (Laird, Newell, and Rosenbloom, 1987) SOAR captures problem solving plans by compiling (chunking) a sequence of rules into a single one called a chunk. The LHS of a chunk consists of the union of the LHS predicates which exist in the rules which were executed, while the RHS of a chunk consists of the last predicate that resulted in the achievement of a problem solving goal.

SOAR distinguishes between two types of knowledge: *search control*, and *task implementation*.

Each type of knowledge can be partitioned into *problem spaces*, *goals*, and *operators* for achieving the goals in each space. In this respect the knowledge organization of SOAR is very similar to that of TLTS, with spaces corresponding to knowledge sources, the notion of the goals is the same in both systems, and the operators corresponding to either redesign operators or analysis and diagnosis rules. Just as in the case of SOAR, TLTS uses two types of knowledge: control (meta-rules, eligibility demons, and scheduling rules), and implementation (analysis rules, diagnosis rules, redesign operators). Each of the types of control knowledge in TLTS can be thought of as belonging to a different space.

The notion of forming new problem solving spaces using operators that are defined in existing spaces is not present is SOAR. The chunks that are being formed represent a piece of a problem solving activity but no further compilation of the knowledge in the chunks takes place. Because of the form the chunks take, the systems produced using SOAR are hard to inspect, modify, and maintain. The developers of SOAR claim that updating or modification of the knowledge base is done by learning new chunks. However, we feel that the systems that have been built thus far using SOAR are not large enough so that knowledge modification and maintenance issues become prevalent.

The PRODIGY system (Minton, 1988) is also similar to TLTS in that it integrates a performance system and a learning component and it separates the search control from the domain or implementation knowledge. The learning component of PRODIGY is used for the acquisition of various types of control knowledge and not of implementation knowledge. In this respect the learning component of PRODIGY is different from the learning component of TLTS. However, in the area of learning control rules, the learning component of PRODIGY learns both rejection and selection rules whereas the learning component of TLTS learns only selection rules.

FUTURE WORK

There are two areas where we will focus our future work.

1. **Redesign operator acquisition.** The fundamental assumption made in this work is that all the operators which are used in the knowledge source being formed by the learning component have already been defined. Even though, as it was explained earlier in this paper, this is a good assumption for the domain of pc board design, we have still encountered cases when it was important to define a new operator. We are looking into high level languages, including graphic description languages, which will allow the designer to define new operators.
2. **Incorporation of domain specific knowledge in control rules.** Currently, the rules which control the execution of the operators within a redesign knowledge source express only precedence relations among the operators. Even though these relations implicitly contain some domain specific knowledge they do not contain all the required knowledge. For this reason we are looking in methods to make more explicit the redesign goals which are being achieved by the operators so that we will be able to capture the domain specific knowledge that is used when selecting which operator(s) will be executed during a particular redesign step. We have been able to do this in the case of the acquiring meta-rules because we were able to take into account the results from the analysis and diagnosis phases.

CONCLUSIONS

The work on the learning component and the study of other learning systems has lead us to conclude that learning is facilitated when the learning component is integrated with a performance system. When doing so, the knowledge that the performance system contains, as well as the generated problem solving information can be directly utilized by the learning component. Furthermore, the quality and effect of the knowledge that is learned can be directly utilized by the performance system and appropriate action can be taken as needed by the learning component.

Since the representation of the knowledge that will be acquired by the learning component has already been defined for the purposes of the performance system, TLTS, no intermediate representations for the acquired knowledge are required. This makes easier the validation and verification of the knowledge that is acquired.

The model that underlies the learning component described here allows the incremental development of a piece of redesign knowledge. The techniques described are similar to the ones which are used by designers when they are determining new ways of modifying faulty circuits. In this respect, in addition to its practical application, the work on the learning component has allowed us to study some of the cognitive aspects of pc board design.

Finally, by compiling the redesign knowledge of only one plan into a knowledge source, including eligibility and operator control knowledge, we are achieving the same results as if were replaying plans. However, if more than one plan is compiled into a knowledge source, the generalization of the operators and the other knowledge that is acquired, gives to the performance system TLTS the capability to form more plans and thus solve more problems than a system which can solve a problem by replaying a plan. In order for the latter system to be able to solve a variety of problems it will have to utilize analogy, which may be difficult to identify in certain domains, including design.

Acknowledgements

I would like to thank Kamesh Ramakrishna for providing many useful suggestions during the development of the project and during the writing of this paper. I would also like to thank: John McDermott, Dave Hartzband, and Meyer Billmers for providing constructive feedback to earlier versions of this paper.

REFERENCES

Acosta, R.D., Huhns, M.N., S-L. Liuh (1986). "Analogical Reasoning for Digital System Synthesis". *Proceedings of IEEE ICCAD*, pp. 173-176, November 1986, Santa Clara, CA.

Carbonell, J., (1983a). "Learning by Analogy: Formulating and Generalizing Plans from Past Experience", in R.S. Michalski, J. Carbonell, and T. Mitchell (Eds.), Machine Learning, pp. 137-161, Palo Alto, CA., Tioga Publishing Company.

Carbonell, J., (1983b). "Derivational Analogy and its Role in Problem Solving", *Proceedings*

AAAI83, pp. 64–69, 1983, Washington, D.C.

Minton, Steven (1988) "Qualitative Results Concerning the Utility of Explanation–Based Learning", *Proceedings AAAI88*, pp. 564–569, August 1988, Saint Paul, MN.

Mostow, J., and Barley, M. (1987). "Automated Reuse of Design Plans", *Proceedings ICED'87*, pp 632–647, August 1987, Boston, MA.

Laird, J.E., Newell, A., and Rosenbloom, P.S. (1987). "Soar: An Architecture for General Intelligence", *Artificial Intelligence*, Vol. 33, No. 1, pp. 1–64, 1987.

Simoudis, E., "A Knowledge Based System for the Evaluation and Redesign of Digital Circuit Networks", *IEEE Transactions in CAD*, forthcoming.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Extracting Lexical Knowledge from Dictionary Text†

Brian M. Slator

Department of Computer Science
North Dakota State University
Fargo, ND 58105
ncslator@plains.nodak.edu

Abstract

A sub-system that produces text-specific lexicons from selected machine-readable dictionary definitions has been developed, as part of a larger system for Preference Semantics and knowledge-based parsing. The input to this sub-system is unconstrained text; the database is Longman's Dictionary of Contemporary English (LDOCE, a machine-readable dictionary which is, itself, simply a special purpose text); the output is a collection of lexical semantic objects, one for every *sense* of every word in the text. Each lexical semantic object in this lexicon contains grammatical and sub-categorization information, often with general (and sometimes specific) grammatical predictions; most of these objects also have semantic selection codes; and many have contextual (pragmatic) knowledge as well. As a natural side-effect of the lexicon construction, a relative contextual score is computed for each object that bears such a code; these scores provide a simple metric for comparing competing word senses for text-specific contextual coherence, and so directly address the problem of lexical ambiguity. Besides exploiting those special encodings supplied with the dictionary entries, the text of selected dictionary definitions are analyzed, to further enrich the resulting representation.

1. Introduction

A lexicon providing sub-system, that produces text-specific lexicons from selected machine-readable dictionary definitions from the Longman Dictionary of Contemporary English (LDOCE; Procter 1978) has been developed (Fig. 1.1). Although designed as part of a larger system for Preference Semantics and knowledge-based parsing, the resulting lexicons are in a very general predicate and frame representation that would have utility as part of a variety of differing systems of language analysis. The input to this sub-system is unconstrained text; the output is a collection of lexical semantic objects, one for every *word sense*, of every part of speech, of every word in the text. Each lexical semantic object in this lexicon contains grammatical and sub-categorization information, often with general (and sometimes specific) grammatical predictions; most of these objects also have semantic selection codes; and many have contextual (pragmatic, LDOCE subject code) knowledge as well. As a natural side-effect of the lexicon construction, a relative contextual score is computed for each object that bears such a pragmatic code; these scores provide a simple metric for comparing competing word senses for text-specific contextual coherence, and so directly address the problem of lexical ambiguity. Besides exploiting those special encodings supplied with the dictionary entries, the text of selected dictionary definitions are analyzed, to further enrich the resulting representation.

The sub-system for constructing the lexicon accesses two files created previously in a preprocessing phase:

1. a file of LDOCE word-sense definitions, in the form of Lisp lists, extracted from the online machine-readable dictionary; and
2. a list of word counts, giving the number of instances of each word in the source text;

and then creates from these a memory structure for use by a Preference Semantics parser (Slator 1988a, 1988c). This lexicon sub-system is implemented as a package in Common Lisp. The packaging structure accords this sub-system its own name-space for the words in the lexicon, and this prevents collision with the Preference Semantics

† This research was supported by the New Mexico State University Computing Research Laboratory — grateful acknowledgement is accorded to Dr. Yorick Wilks and all the members of the CRL Natural Language Group for their continuing interest and support.

parser which operates over orthographically identical symbols for words in the text.

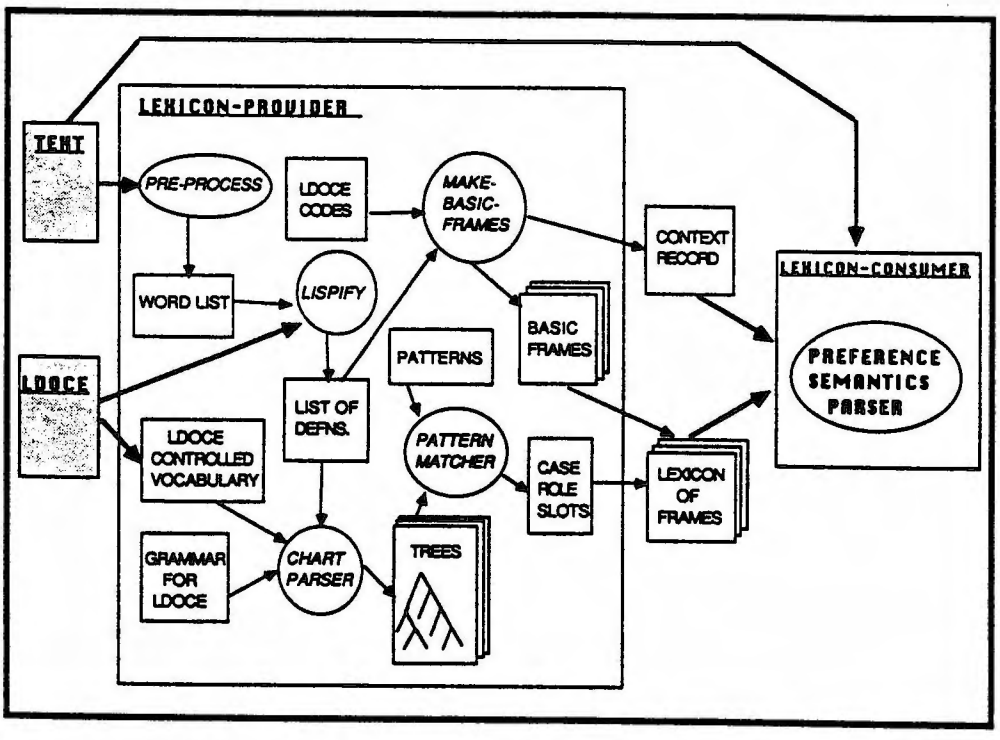


Fig. 1.1: Lexicon Sub-System Overview

This lexicon constructing sub-system performs three distinct functions.

1. Make Basic Frames — this mostly entails string manipulations and table lookup, to transform the definition list into a frame-like structure.
2. Build a Context Structure — as the definitions of the words of the text are being transformed into basic frames a record is kept of all the pragmatic codes that are used; this record, in conjunction with the wordcount file created during the text manipulation and preprocessing phase, is used to fill in a specially restructured hierarchy of pragmatic codes. Once the appropriate values have been propagated through the hierarchy this structure is used by the Preference Semantics parser to prefer certain word senses over others. This provides a global coherence mechanism that mediates against the naturally local semantic and grammatical propensities of the preference mechanism.
3. Enrich the Frames — further lexical information is available within the text of the definitions. This is extracted by parsing the definition text (with a chart parser extensively modified from an algorithm published in Slocum (1985)), and then, by pattern matching, extracting labeled slots from the resulting parse tree as *genus* and *differentia*.

1.1. Motivation

All of the, by now, familiar arguments for natural language systems based on existing machine-readable knowledge sources, need not be repeated here (Amsler 1986; Markowitz, Ahlswede and Evens, 1986; Binot and Jensen 1987). Anyone who has, or has thought about, building a realistic lexicon or other knowledge resource *by hand*, knows or can imagine what a daunting prospect this is.

1.2. Economies of Scale

George Miller (1985) makes the following observation:

> ... there seem to be two approaches to the vocabulary problem. One uses a machine-readable version of some traditional dictionary and tries to adapt it to the needs of a language processing system. Call this the *book* approach. The other writes lexical entries for some fragment of the English lexicon, but formulates those entries in a notation that is convenient for computational manipulation. Call this the *demo* approach. The *book* approach has ... [many] words ... [but] each is difficult to use. The *demo* approach has ... easy to use [words] but ... usually not many.

In an unpublished manuscript, Bran Boguraev relates the following anecdote:

> At a recent workshop[1] an informal poll was conducted among the invited speakers to establish the average size of the lexicons their respective systems were using. Taking into account a five to six thousand word ... machine translation system, ... the average ... came to 1500 words. Without this [system] the average was about 250.

Taken together, these two quotes revealingly outline the state of the art in Natural Language Processing (NLP) today. In short, there are two ways to build a lexicon: by hand or by program -- and everyone does it by hand. This is not to say that remarkable performance cannot be achieved with a small lexicon. On the contrary, virtually all of the successful language processing systems have operated over relatively few words. For example, Terry Winograd has confirmed (personal communication) that SHRDLU, the "blocks world" interface and arguably the most famous language processing program on record, had a vocabulary of only about 200 items.

As just outlined, in the world of computational language understanding there are two methods of lexicon building: the *book* approach and the *demo* approach (Miller 1985). The *demo* approach is to encode, by hand, a small but rich lexicon suitable for the analysis of a (sometimes considerable) fragment of language, usually with a view to explaining a theory of specific language phenomena such as metaphor or metonymy. Characteristically, these systems of analysis have a representation that is easy to use, being designed with a specific purpose in mind; however, systems of this sort suffer well-known difficulties in "scaling up." Nonetheless, this has been the dominant paradigm in natural language processing (and in AI in general), for the last two decades.

The *book* approach begins with a machine-readable language resource (such as Webster's Seventh), and attempts to extract the necessary linguistic knowledge in a suitable form, usually with a view to covering as large a portion of the language as possible. Characteristically, these systems of analysis have a representation that is difficult to use, being extracted from sources designed for other purposes; however, systems of this sort have the advantage of built-in size and principled lexical knowledge constructed through countless years of effort by professional lexicographers. Nevertheless, it is hard to over-state the difficulties of extracting necessary and sufficient information from machine-readable sources intended for human, and not computer, consumption.

The fundamental differences between the two approaches are easily contrasted. The demo-lexicon-producer handcrafts lexical entries with fore-knowledge of what they will be used for, and what they should contain; these entries will be complete for their intended purposes, but they will be "expensive" to produce. The demo-lexicon-consumer has the luxury of working with lexical objects that are ideal, but few in number — this person can concentrate on the development of the (lexicon-consuming) language processing system itself, within the limits of the lexicon.

Conversely, the book-lexicon-producer must first develop a system for analyzing the machine-readable resource and extracting as much information as possible from it; this is no easy task, particularly if the language resource is as linguistically difficult, or nearly so, as the language itself. However, once this is accomplished the lexical entries will be relatively "cheap." The book-lexicon-consumer has the luxury of working from a large knowledge base, but one made up of less-than-ideal entries — this person must concentrate on making the most of what they have. A central issue, then, is whether it is possible in principle, and in medically finite time, to produce a lexicon that is both rich enough to be useful to a language understanding system and big enough to cover a significant subset of the language.

Longman's Dictionary of Contemporary English (LDOCE; Procter et al. 1978), is a full-sized dictionary designed for learners of English as a second language that contains over 55,000 entries in book form and 41,100 entries in

[1] The 1986 Alvey/ICL Workshop on Linguistic Theory and Computer Applications.

machine-readable form (a type-setting tape). The book and tape versions of LDOCE both use a system of grammatical codes of about 110 syntactic categories which vary in generality from, for example, *noun* to *noun/count* to *noun/count/followed-by-infinitive-with-TO*. The machine readable version of LDOCE also contains "box" and "subject" codes that are not found in the book. The box codes use a set of primitives such as **abstract, concrete,** and **animate,** organized into a type hierarchy. These primitives are used to assign type restrictions on nouns and adjectives, and type restrictions on the arguments of verbs. The subject codes, referred to here as "pragmatic" codes to avoid confusion with the grammatical subject, are another set of terms organized into a hierarchy. This hierarchy consists of main headings such as **engineering** with subheadings like **electrical.** These terms are used to classify words by subject, for example, one sense of *current* is classified as **geology-and-geography** while another sense is marked **engineering/electrical.** LDOCE is a relatively untapped resource; and to this is applied a prototypical *book* approach. LDOCE is a full-sized dictionary in machine-readable form, designed for learners of English as a second language, and containing several non-standard features (grammar, type, and pragmatic codes). The lexicon-providing sub-system takes LDOCE as a database and provides a structured (and much smaller) knowledge base of lexical semantic objects organized by pragmatic context.

1.3. Knowledge Based Parsing

The knowledge base constructed by the lexicon-provider sub-system is used by a Preference Semantics parser (Wilks 1972, 1975a, 1975b, 1978; Boguraev 1979; Carter 1984, 1987; Fass 1986, 1987, 1988; Huang 1984, 1988; Slator 1988a, 1988c). Preference Semantics is a theory of language in which the meaning for a text is represented by a complex semantic structure that is built up out of smaller semantic components; this compositionality is a fairly typical feature of semantic theories. Preference semantics is principally distinguished from other semantic theories by its explicit and computational accounting of ambiguous, metaphorical, and non-standard language use. Knowledge is required to compute preferences for deciding or resolving lexical ambiguity, constituent attachment (structural ambiguity), and referential ambiguity, and LDOCE can be a rich source of such knowledge. The Preference Semantics parser receives two inputs: a text to be parsed, and a lexicon of semantic objects specific to that text. Here is described a lexicon-providing sub-system for *creating* the necessary lexical semantic knowledge base, preliminary to Preference Semantics parsing; this sub-system also has two inputs: the text being parsed, and LDOCE.

1.4. Background

As a technical term "Lexical Semantics" is in a state of flux, and is taking on added meaning. While once reserved to a sub-area within descriptive linguistics, the term has lately come to apply to machine-readable dictionary analysis, at least insofar as the product of that analysis is some arguably semantic artifact. Originally part of the attempt to place lexical items within the context of their unique "semantic field" -- that is, roughly, the range of contexts in which a given lexical item has a particular semantic import; Lexical Semantics has also become an attempt to construct meaningful objects out of more-or-less primitive semantic components,[2] within their particular textual (dictionary) context.

These two sense descriptions of Lexical Semantics seem orthogonal at first glance, and in fact they are, at least in terms of processing. A linguist, when analyzing a particular lexical item looks at patterns of use, co-occurrence, and collocation, finding those contexts where the lexical item is "welcome," and finding other items that can substitute for the original. This is done by inspecting and analyzing corpora or doing other forms of linguistic field-work. The result of an analysis of this kind is a linguistic description of a lexical item in terms of its use. This linguistic description is part of what a lexicographer needs to know in order to decide in which events a lexical item has taken a different sense; that is, which uses of a lexical item deserve their own sense definition.

These definitions, as encoded into the special defining sub-language of the dictionary by lexicographers, by reference to the semantic fields of the linguist, are the grist of Lexical Semantics in its new sense. The analysis of machine-readable dictionary entries attempts to "unpack" the various pieces of encoded information stored there by the lexicographer. This construction implies a connection between descriptive linguists, and their lexical semantic products, and the data professional lexicographers use to build dictionaries, but this connection probably does not literally exist. Lexicographers have their own tools and resources. The lexical semantic information, or a part of it, is nonetheless encoded in dictionaries and the difference is that it is accessible there, to machine analysis, where it is

[2] "More-or-less primitive" because the contents of lexical semantic frames, as extracted from the dictionary, are a mixture of primitive elements and partially disambiguated controlled vocabulary items.

not in the original corpora of the linguist. This new sense of Lexical Semantics is complementary to the original sense, because the focus is not discovering lexical knowledge, but rather on recovering lexical knowledge discovered and encoded by others.

A dictionary is a text whose subject matter is language. The purpose of dictionaries is to provide definitions of senses of words and, in so doing, supply knowledge about not just language, but the world. Researchers in Computational Linguistics (CL) and Artificial Intelligence (AI) have viewed dictionaries:

1. with theoretical interest as a means of investigating the semantic structure of natural language; and
2. with practical interest as a resource for overcoming the knowledge acquisition bottleneck in AI: how to acquire formal meaning and knowledge representations automatically.

A dictionary is a text with certain special features. First, dictionary entries are both condensed and incomplete: the writer of a dictionary makes assumptions about the knowledge brought to bear by the reader of a dictionary. Second, dictionary entries are written in a special sublanguage as a result of these assumptions: the grammar of dictionary definitions is different from the grammar of general English. Further, the sublanguage of LDOCE is unique with respect to most other dictionaries due to the use of a "controlled vocabulary" within the definitions: this is a mixed blessing since the limited vocabulary reduces the size of the dictionaries internal lexicon at the expense of using these vocabulary items in more than a minimum number of senses and in sometimes unnatural and verbose ways. In sum, given the non-contentious assumption that there is knowledge encoded in dictionaries, and the further assumption that this knowledge can be extracted by programs, there still remains several obstacles to clear. However, it is a reasonable limiting assumption, and a fundamental assumption of this research, that the problem of dictionary analysis is easier than the general problem of language understanding by computer and, further, that workable solutions to this first problem can be seen as steps towards that still distant goal.

2. Constructing Contextually Organized Lexical Semantic Knowledge-Bases

The lexicon-providing subsystem is principally composed of programs for constructing lexical semantic knowledge bases. These programs take online machine-readable dictionary entries and first produce corresponding basic lexical semantic objects (see Fig. 2.2). This much is accomplished in a straight-forward way and is, by itself, a useful and consistent knowledge base for parsing (Wilks, Fass, Guo, McDonald, Plate, and Slator, 1987, 1988). These basic frame-objects are further manipulated in two ways:

1. they are organized into a hierarchical contextual structure and
2. they are *enriched* by means of further processing over the text of the dictionary definitions themselves, as described below.

Consider, for example, the following short text:

(1) *Current can be measured.*
The technician measures alternating current with an ammeter.

The basic lexicon for this text contains 30 frames for content words (not counting 18 senses of the infinitive *be* and 10 senses of the auxiliary *can*). These 30 frames are: *alternate* (3 adjective senses, 1 verb sense), *ammeter* (1 noun sense), *can* (2 nouns, 2 verbs), *current* (3 adjectives, 4 nouns), *measure* (8 nouns, 3 verbs, 1 adjective), *technician* (1 noun sense), and the phrase *alternating current* (1 noun sense). LDOCE defines about 7,000 phrases: from *Achilles tendon* and *American Football* to *yellow press* and *zoom lens*.

Each basic frame has five slots: GRAMMAR, POS, SENSE-NUM, PRAGMATIC, and TYPE (as in Fig. 2.2). The GRAMMAR slots are filled with category information and predictions from the LDOCE grammar for English with its 110-odd members (Quirk et al., 1972, 1985); POS is part of speech, the top level of the GRAMMAR hierarchy; and SENSE-NUM is the sense number. PRAGMATIC slots are filled with contextual domain terms, like **engineering** or **occupations.** TYPE slots are filled, in the case of nouns and adjectives, with selection restrictions like **human, abstract,** or **solid,** and in the case of verbs, with selection restrictions on the functional arguments to the verb such as **human** Subject and **abstract** Direct Object.

```
(technician
 (POS . n)
 (SENSE-NUM . 0)
 (GRAMMAR
  ((C) . N/Count)))
 (PRAGMATIC
  (ON (Occupations)))
 (TYPE
  ((H) (RESTRICTION . Human/Sex-Unspecified)))
(current
 (POS . n)
 (SENSE-NUM . 3)
 (GRAMMAR
  ((C) . N/Count))
 (PRAGMATIC
  (EGZE (Engineering/Electrical)))
 (TYPE
  ((T) (RESTRICTION . Abstract))))
(measure
 (POS . v)
 (SENSE-NUM . 1)
 (GRAMMAR
  (or
   ((I) . V/Intransitive)
   ((T1) . V/Transitive/N+-or-PRON+-Follows)))
 (PRAGMATIC nil)
 (TYPE
  ((H T -)
   (SUBJECT . Human/Sex-Unspecified)
   (OBJECT1 . Abstract)
   (OBJECT2))))
(ammeter
 (POS . n)
 (SENSE-NUM . 0)
 (GRAMMAR
  ((C) . N/Count)))
 (PRAGMATIC
  (EGZE (Engineering/Electrical)))
 (TYPE
  ((J) (RESTRICTION . Solid/Movable)))
```

Fig. 2.2: Basic Lexical Semantic Objects (Frames for text 1).

2.1. Contextual Preference

Consider the string

(2) *"The band saw and the old felt ..."*

Without knowing what follows in the ellipsis (as must be the case when the text is read for the first time), and without any surrounding context, the correct phrase boundaries in (2) are undecidable. A reader will probably tend to see a pair of conjoined noun phrases (NPs) (concerning a woodworking machine and some aged fabric), waiting for a verb phrase (VP) to appear in the ellipsis; but with a little suitable priming might also easily see a pair of conjoined transitive sentences (Ss) (about a musical ensemble and a group of senior citizens), waiting for a direct object NP. The two possible segmentations are these:

(i) [and [NP [The band saw]] [NP [the old felt]]] ...

(ii) [and [S [NP The band] [VP saw]] [S [NP the old] [VP felt]]] ...

The problem here is really one of category selection. Are *saw* and *felt* nouns or verbs? The point is not that a continuation of

a. ... *were sold on Friday*

selects the first segmentation while a continuation of

b. ... *an explosion on Friday*

selects the second. The point is that one or the other of these choices can be preferred at the moment it is encountered, even if this sentence is the first in the text, even without seeing the continuation. With pragmatic codes precomputed for the text in question, as a context setting mechanism, the correct word sense is strongly preferred, hence the correct category.

Again, consider the text:

(1) *Current can be measured.*
The technician measures alternating current with an ammeter.

Without context the correct sense of *current* in the first sentence cannot be selected until after the second sentence is processed; and with context, a strong candidate can be preferred. It would be natural to expect this process to work better on longer text, the longer the better, and indeed this appears to be true in the tests that have been run. However, good results can be had even with these relatively short fragments.

The sub-system for constructing the lexicon also establishes the conceptual domain of texts. In the basic lexicon construction process, all of the word senses, for all parts of speech, of all of the words in the text are looked up, giving a "text-specific" lexicon. Along the way, an ordered list of pragmatic (LDOCE, subject) codes is collected for "content assessment" (Walker and Amsler 1986, where lists of frequently occurring LDOCE pragmatic codes are compiled by simply counting up the instances found in the various senses of the words in the text). The LDOCE pragmatic coding system divides the world up into 124 major subject categories ranging from **aeronautics, aerospace,** and **agriculture,** through **geology-and-geography, glass,** and **golf,** to **vehicles, water-** and **winter-sports,** and **zoology.** Many of these subjects are further subcategorized (for example, under **agriculture** is **soil-science** and **horticulture,** and under **zoology** is **entomology, ornithology,** and **ichthyology**), so there is a total of 369 different subject codes in the LDOCE pragmatic system. However, the LDOCE hierarchy is flat (only two layers deep), and the 124 major categories have equal and unrelated status; for example, **business** and **economics** are both at the top of the tree and are unconnected, the same is true of **science** and **zoology** (and, for other examples, see Fig. 2.3).

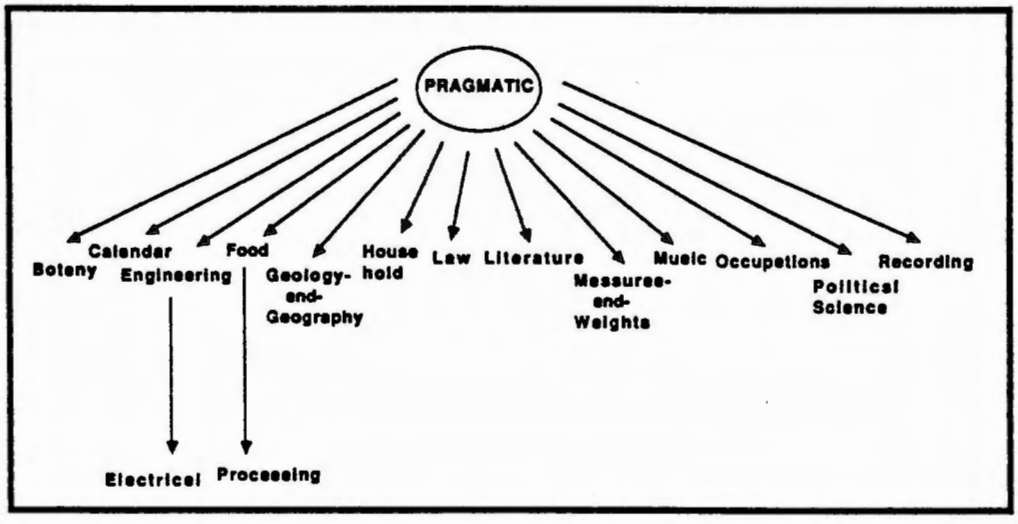


Fig. 2.3: A Portion of the Flat LDOCE Pragmatic Hierarchy

The scheme implemented here first selects the relevant portions of the flat LDOCE hierarchy, with enumerated text elements at the leaves (Fig. 2.4), and then imposes deeper structure onto the LDOCE pragmatic world (Fig. 2.6), relating pragmatic categories in a natural way, in order to discover important relationships between concepts within text. This particular restructuring is not one to be defended point by point; there has been, for instance, an arbitrary division made at the highest level. What can be defended is the notion that, for example, words classified under **botany** have pragmatic connections to words classified as **plant-names,** as well as connections with other words classified under **science** (connections *not* made by the LDOCE pragmatic hierarchy as given), and that these connections are useful to exploit when attempting to determine the subject matter of a text, or when attempting to choose the correct sense of polysemous words.

The lexicon-providing sub-system relates these codes through a specially restructured hierarchy created for that purpose, making **communication, economics, entertainment, household, politics, science,** and **transportation** the fundamental categories. Every word sense defined with a pragmatic code therefore has a position in the new hierarchy, attached below the node for its pragmatic code. Every node in the hierarchy is assigned a value by the lexicon-provider according to the number of words in the original text that bear that code, and then values lower in the structure are propagated by summing upwards towards the root. At the end of this process a single pragmatic code for one of the seven fundamental categories, high in the hierarchy and therefore general in nature, asserts itself as the domain description term for the text. The result is a structure for lexical ambiguity resolution because this process also yields a set of preferred word senses that cohere with the subject domain of the text (in the sense that a set of, say, engineering terms are coherent with respect to engineering). The implication of discovering a global domain

description term for a text is that this term carries an entire sub-hierarchy of more specific pragmatic codes with it. Every word sense defined with a pragmatic code therefore has a position in the hierarchy, attached to the node for its pragmatic code, with the scores computed at every relevant node. The Preference Semantics parsing algorithm, described in Slator (1988a, 1988c), strongly prefers those word senses coherent with, i.e. in the same subject area with, the domain description term established for the text: the more specific the word sense (the lower in the restructured pragmatic hierarchy), the higher the preference.

In the table of Fig. 2.5, the **Pre-Restructuring** column is compiled by simply counting up the LDOCE pragmatic codes found in the various senses of the words in text 1 (as in "content assessment" described above). The **Post-Restructuring** column is the scores resulting from the deeper pragmatic structure. Note that domain descriptors like **economics** and **science** have been introduced, and that the status of **science** as a global domain descriptor for the text only asserts itself after restructuring. Beforehand, **science** is not on the list and **engineering/electrical, measures-and-weights,** etc. are of equal and unrelated status. This is clearly an over-compartmentalized view of the world. The deeper hierarchy gives a far better intuitive ordering of the important concepts in each text than the given LDOCE hierarchy.

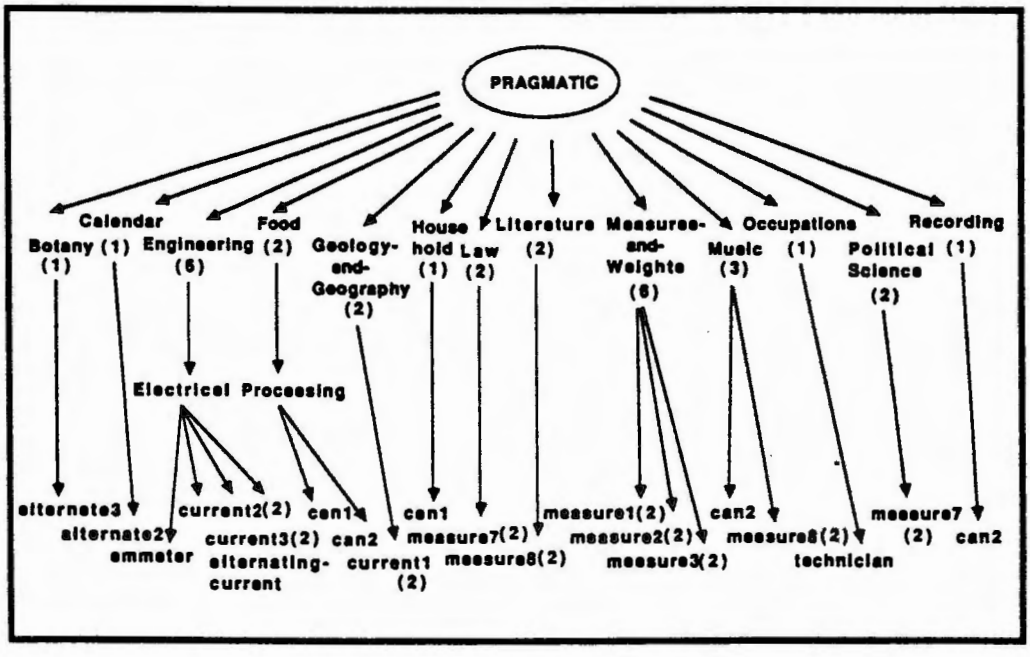


Fig. 2.4: The LDOCE Pragmatic Hierarchy (with words from text 1)
The parenthetical numbers are from the Pre- column of Fig. 2.5

2.2. Processing Details

The restructured pragmatic hierarchy is a data structure that has values inserted at leaf nodes according to how many times a specific code is used in different word-sense definitions, multiplied by the number of occurrences of those words in the text. For example, the word *ammeter* has a pragmatic code marking **engineering/electrical** and if *ammeter* occurs five times in the text then the node for **engineering/electrical** is given a score of five. If another word, say *current* also has an **engineering/electrical** sense, and *current* is used eleven times in the text, then the score for the **engineering/electrical** node is increased to sixteen (five plus eleven). Then, if another word in the text, say *levee* has a pragmatic code marking **engineering/civil** and *levee* appears four times in the text, then **engineering/civil** will receive a score of four and **engineering,** the parent of these two, will receive a score of twenty (sixteen plus four, the sum of both children, **electrical** and **civil**). Values are propagated upwards in the hierarchy in this manner, and at the end of this sequence one of the seven or eight most general terms has the highest value and is chosen as a topic description term for the entire text.

Code	Pre-Restructuring	*Score*	Code	Post-Restructuring	*Score*
MS	measures-and-weights	6	SI	science	16
EGZE	engineering/electrical	6	CM	communications	6
MU	music	3	MS	measures-and-weights	6
PL	political-science	2	EG	engineering	6
LT	literature	2	EGZE	engineering/electrical	6
LW	law	2	AF	art	5
GO	geology-and-geography	2	PL	political-science	4
FOZP	food/food-processing	2	MU	music	3
ON	occupations	1	HH	household	3
RE	recording	1	LT	literature	2
HH	household	1	LW	law	2
BO	botany	1	GO	geology-and-geography	2
CA	calendar	1	FO	food	2
	...		FOZP	food/food-processing	2
			EC	economics	1
			BZ	business	1
			ON	occupations	1
			RE	recording	1
			BO	botany	1
			HR	horology-clocks	1
			CA	calendar	1

Fig. 2.5: Pragmatic Codes and Domain Descriptors for Text 1.

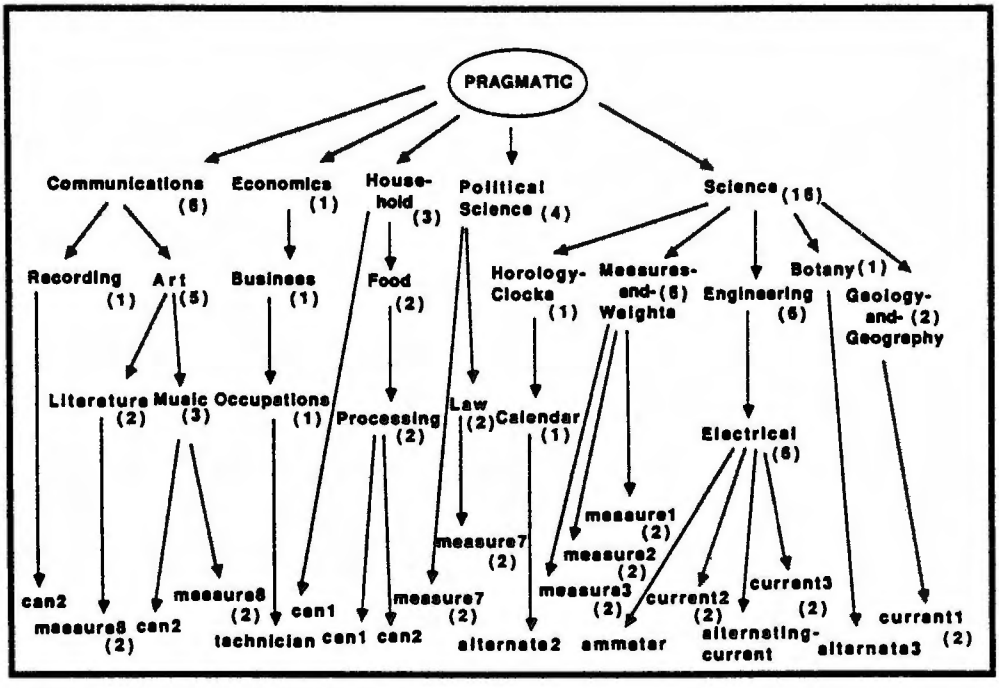


Fig. 2.6: The Restructured LDOCE Pragmatic Hierarchy (with words from text 1)
The parenthetical numbers are from the Post- column of Fig. 2.5.

Thereafter, words are given contextual scores relative to their position in the hierarchy. In the current implementation, a word is given a contextual score equal to the sum of the scores on the path from that word to the root of the pragmatic hierarchy divided by the value of the domain description term: yielding a value between 0 and 2.0. In this way more specific (deeper) word senses are given higher scores; for example, in Fig. 2.6 the geology-and-geography sense of *current* (current1), receives a contextual score of 20 for text 1, the sum of the scores on the path

from there to the root:

current 1(2)+*geology–and–geography* (2)+*science* (16)=20/*science* (16)=1.25

while the **engineering/electrical** senses of *current* (current2 and current3), both receive scores of 30:

current 2(2)+*electrical* (6)+*engineering* (6)+*science* (16)=30/*science* (16)=1.875

Some other mechanism must choose among the word senses with identical scores, using other sources of information, and the Preference Semantics parser (Slator 1988a, 1988c), does this, but the contextual preference mechanism still achieves a significant result by itself by preferring the two **engineering/electrical** senses of *current* over the other five possibilities. This pragmatic hierarchy structure captures a notion of text topic, or context, within which framework word sense decisions are later made by the Preference Semantics parser.

2.3. Enriching Frames

The basic frames of Fig. 2.2 are a large step towards the knowledge required for Preference Semantics parsing - by a lexicon-consumer operating over non-dictionary text. However, there is a hidden wealth of further information within the text of the definitions themselves. When the needs of a knowledge-based parser increase beyond this initial basic representation (as is the case whenever, say, resolving lexical ambiguity or making non-trivial attachment decisions), the frame representations are enriched by appeal to parse trees constructed from the dictionary entries of the relevant word senses. This additional information further enriches the semantic structures.

Dictionary definitions have been characterized (Amsler 1981), as being composed of *genus* and *differentia* terms. The *genus* term is an "upward-pointing" reference (or hypernym), that places a word within a superordinate class; and they are essential for distinguishing between members of a primitive TYPE. For example *ammeter* and *automobile* are both of TYPE **solid**, but the *genus* of the first is *instrument* while the *genus* of the second is *vehicle*. In effect, *genus* terms introduce intermediate levels into a TYPE hierarchy. On the other hand, *differentia* terms are a means of distinguishing among elements within a *genus*. For example, *thermometer* and *chronometer* are both of *genus instrument*, (which is itself of TYPE **solid**), but they are differentiable because one measures heat while the other measures time.

A chart parser has been developed that accepts LDOCE definitions as Lisp lists and produces phrase-structure trees. LDOCE sense definitions are typically one or more independent clauses composed of zero or more prepositional phrases, noun phrases, and/or relative clauses. The syntax of sense definitions is relatively uniform, and developing a grammar for the bulk of LDOCE has not proven to be an intractable problem. Chart parsing was selected for this system because of its utility as a grammar testing and development tool. The chart parser is driven by a context free grammar of 100-plus rules and has a lexicon derived from the 2,219 words in the LDOCE core vocabulary. The parser is left-corner, and bottom-up, with top-down filtering (taken from Slocum 1985), producing phrase-structure trees.

The context-free grammar driving the chart parser is virtually unaugmented and, with certain minor exceptions, no procedure associates constituents with what they modify. Hence, there is little or no motivation for assigning elaborate or competing syntactic structures, since the choice of one over the other has no semantic consequence (Pulman 1983). Therefore, the trees are constructed to be as "flat" as possible. The parser also has a "longest string" (fewest constituents) syntactic preference. The grammar is still being tuned, but the chart parser is already quite successful and works extremely well over a fairly wide range of examples from the language of content word definitions in LDOCE (about 1200 so far). Ninety-Five percent result in a parse tree for the entire definition text. Five percent of the analyses fail at some point. In those cases where it fails the parser still returns a partial parse (of the leading constituents in the definition text), and this is the most important part of a definition anyway. This chart parser is not, it must be emphasized, a parser for English — it is a parser for the sub-language of LDOCE definitions (Longmanese), and in fact only the open class (content word) portions of that language. A tree interpreter extracts semantic knowledge from these phrase-structure definition trees (Slator 1988a, 1988b; Slator and Wilks, 1987); some examples of chart parser trees appear in Fig. 2.7 in a cleaned up form.

2.3.1. The Tree Interpreter

The output of the chart parser, a phrase structure tree, is passed to an interpreter for pattern matching and inferencing. The first step of tree interpretation is to pick off the dominating phrase and, after restructuring it into GENUS and FEATURE components (by reference to the currently active grammar version), insert it into the current basic frame under a GENUS slot. Further strategies for pattern-matching are being developed, to extract more detailed differentia information, beyond the GENUS and its FEATURE modifiers.

```
(technician
  (NP
    (DET . a)
    (Mods
      (ADV . highly)
      (ADJ . skilled)
      (ADJ . scientific)
      (CONJ . or)
      (ADJ . industrial))
    (N . worker))
  (SEP . ;)
  (NP
    (N . specialist))
  (PP
    (PREP NP)
     in
     (NP
       (DET . the)
       (ADJ . practical)
       (N . detail)))
  (PP
    (PREP NP)
     of
     (NP
       (DET . a)
       (N . subject))))
```

```
(current
  (NP
    (DET . the)
    (N . flow))
  (PP
    (PREP NP)
     of
     (NP
       (N . electricity)))
  (PP
    (PREP NP)
     past
     (NP
       (DET . a)
       (ADJ . fixed)
       (N . point))))
```

```
(ammeter
  (NP
    (DET . an)
    (N . instrument))
  (PP
    (PREP NP)
     for
     (NP
       (N . measuring)))
  (COM . ,)
  (PP
    (PREP NP)
     in
     (NP
       (N . amperes)))
  (COM . ,)
  (NP
    (DET . the)
    (N . strength))
  (PP
    (PREP NP)
     of
     (NP
       (DET . an)
       (ADJ . electric)
       (N . current))))
```

Fig. 2.7: Phrase-Structure Trees from the LDOCE Chart Parser.

The relationship between a word and its definition can trivially be viewed as an "IS-A" relation; for example, an *ammeter* is *an instrument for measuring ... electric current*. The frame created for each word-sense from its definition, then, represents the *intension* of that word-sense. This observation motivates the assumption that portions of this intensional material can be isolated and given a label for eventual preference matching. For example, by noting that an *ammeter* is *for measuring* it becomes reasonable to create a slot in the *ammeter* frame that is labeled PURPOSE and filled with *measuring*. This kind of knowledge is precisely what is needed to compute case roles and preferences (Fig. 2.8).

Consider once again, this sentence from text (1)

(3) *The technician measures alternating current with an ammeter.*

The basic frames (shown in Fig. 2.2), provide information for matching between the SUBJECT slot of *measure* and the TYPE slot of *technician* (both **human/sex-unspecified**), and between the OBJECT slot of *measure* and the TYPE slot of *current* (both **abstract**). And matching against the PRAGMATIC slot of *ammeter* (**engineering/electrical**) selects the preferred sense of *current* (**engineering/electrical** not **geology-and-geography**). However, there is nothing in these basic frames to direct the attachment of *with an ammeter*.

Unfortunately, there is little in the LDOCE definition of *measure* to indicate that it prefers an INSTRUMENT. The LDOCE verbs typically do not predict the case slots they prefer to have filled, as opposed to the situation often found with other language processing systems, rather the nouns can specify case roles that they prefer to fill with (and prepositions can predict the case roles of their noun phrase constituents).

The pattern "**for <verb>-ing**" in the *differentia* of a noun strongly predicts a PURPOSE case role for that noun, and the preposition *with* in English predicts various case roles for its noun phrase object, principally ACCOMPANIMENT (*a man ate a meal with a friend*), POSSESSION (*a man ate a fish with many bones*), and INSTRUMENT. In this instance, the definition for *ammeter* contains a pattern indicating it is a constituent that prefers to fill a

PURPOSE and, in particular, prefers to be *for measuring*. In cases like this a parser should prefer the INSTRUMENT reading over both the ACCOMPANIMENT case relation and the POSSESSION noun phrase complement alternatives because the object of the *with* preposition (*ammeter*), has a PURPOSE case role marker (filled with *measuring*), that suggests an INSTRUMENT attachment. Other case roles that appear extractable from LDOCE differentia, but still subject to further investigation, include PART-OF, MEMBER-OF, MEANS-OF, and MEASURE-OF, as well as some others.

Again, it is important to distinguish this, prior, chart-parser of dictionary texts from the knowledge-based parser of general texts. The output of the former (the lexicon-provider) is the knowledge-base for the latter (the lexicon-consumer). The output of the lexicon-providing dictionary parsing program is a lexicon of word-sense frames, with each frame explicitly or implicitly positioned in multiple, pre-existing, hierarchies.

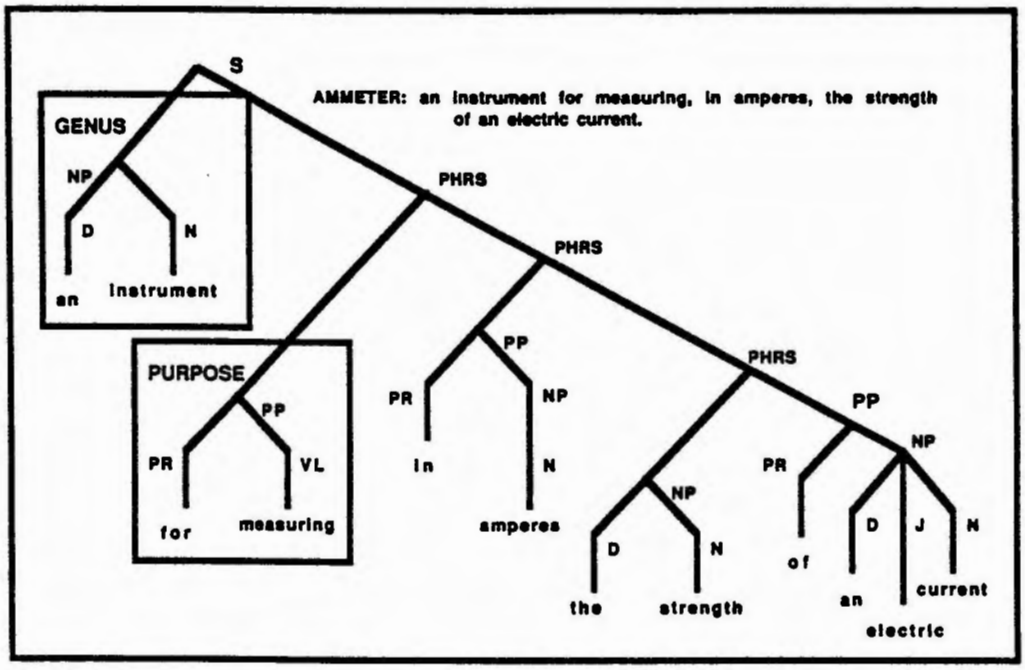


Fig. 2.8: Pattern Matching Against Tree Constituents

2.3.2. Processing Details

Although originally envisioned as a mechanism for enriching an entire lexicon of frames, the parsing and pattern-matching procedure is now only selectively applied. The computational overhead of the enriching procedures is non-trivial and there are many instances where the results are not worth the investment; for example, when the word being defined is a linguistic "function" word the text of the definition is often next to vacuous. In other cases the Preference Semantics parser is quite able to deal with the word simply on the basis of the information in the basic frame (which is constructed with relatively little computational expense). The principle of "least effort" is therefore now operant, the frame enriching procedures of parsing and pattern matching are only called on by the demands of the Preference Semantics parser.

The chart parser is bottom-up with top-down filtering, an approximation to Earley's algorithm. The grammar is small, context-free and unaugmented, but it is also ambiguous and some definition texts produce several trees. A "longest string" syntactic preference dispenses with most of these, and a tree is randomly selected from those that remain. The lexicon for the chart parser is a complete list of all the words that are routinely used in LDOCE definitions, that is, all the members of the LDOCE "controlled vocabulary" and all of their morphological permutations. This vocabulary was constructed as follows.

1. First the 2000+ "core" words listed in the back of LDOCE were typed in. About one-third of these have a single syntactic category specified for them on that list, another third specify more than one category, and the final third are not specified at all (meaning that all relevant categories apply).
2. Categories for the unspecified words were extracted from LDOCE, under program (shell-script) control, by isolating the category-less words and submitting them to an online look-up program. The "core" wordlist is actually nearly 3000 entries after all this.
3. Then this list was augmented by a complete lexeme list (obtained from Tony Plate, see Wilks, Fass, Guo, McDonald, Plate and Slator, 1987, 1988), that contains over 10,000 members; the categories were assigned to each of these new elements by hand (in lieu of implementing a morphological processor).

The parser recognizes cross-references to other definitions (these are either in upper case orthography, or have a trailing sense and homograph number, or both), and treats these words like an unknown word, the grammar allowing for them to be handled like open-class content words. Each lexicon entry is stored in the form required by the chart parser; for example, the noun *house* is represented:

(N (house))

When the *genus* term of a definition text is isolated, there is still a question as to what sense of the constituent head is actually intended. While many of the words in the LDOCE "controlled vocabulary" are used in only a single sense (Cheng-Ming Guo, personal communication), a greater number are used in two or more senses. For example, an *ammeter* is an *instrument*, but there are two defining senses of *instrument* used in LDOCE: *instrument* as a tool of some kind, and *instrument* as a music producer. A scheme has been worked out, but not yet implemented, for looking up the definition of the *genus* term and then selecting the correct one on the basis of matching the semantic and pragmatic codes of the *genus* term with the codes associated with the senses of its definition. There are two easily foreseeable problems with this:

1. the semantic codes are often too general, and the pragmatic codes are too often missing, for this scheme to be completely effective; and
2. the program code for this will be slow, requiring an operating system context switch, which makes it cumbersome to "jump out" and access the dictionary at will.

So this scheme is, at best, a partial solution to the problem.

The tree pattern-matching procedure uses some simple searching and filtering to optimize and avoid unnecessary matching, then a simple theorem-prover is applied (extensively modified from Barr and Feigenbaum, 1981): the patterns are treated as theorems to be proved, and the parse tree is treated as a database. The theorem-prover returns the piece of tree constituent that matches the pattern.

3. Comparison to Other Work

The most closely related recent pragmatic work, and that from which the ideas for this scheme arose, is by Walker and Amsler (1986), who used the LDOCE pragmatic (subject) coding system to do "content assessment" on wire service stories from the New York Times News Service. The focus of that work was to arrive at a list of codes that, taken together, would reflect the subject matter of each story. Their work differs from this scheme in that they accepted the LDOCE pragmatic hierarchy as given, getting good results because their texts, while being arguably general, were highly coherent by nature.

The pragmatic hierarchy scheme has roots in a far older tradition, one that dates to the days of "mechanical translation." Wilks (1972) describes a system for word sense disambiguation used by the Cambridge Language Research Unit (Masterman 1957). Their idea was to choose the correct senses for the words in a sentence by looking them all up in a thesaurus, finding the thesaural heads for each, and then comparing to see which thesaural heads were held in common. Then the sets of words found under the common thesaural heads for each word were themselves compared for commonality, and any that were in the intersection of these sets became sense descriptors for that particular instance of the word in the sentence. This method worked remarkably well, and its major shortcoming was that it failed to take into account word usage that was other than the most "coherent"; therefore, for example, the only sense of *mass* discovered in a physics text would be "mass as weight" and a phrase like *a mass of data* in the same text might have the wrong sense of *mass* assigned to it (Sparck-Jones, 1964, 1986). The pragmatic hierarchy scheme is essentially equivalent to this older idea (although their thesaurus, like LDOCE as given, was only two layers deep and so was not well able to discriminate between competing senses).

Little machine-readable dictionary work has focused on the machine-readable dictionary as a language resource for semantic, knowledge-based parsing. Approaches to extracting semantic knowledge from machine-readable sources, such as "sprouting" (Chodorow, Byrd, and Heidorn, 1985), or the employ of paid "disambiguators" (Amsler and White, 1979; Amsler 1980), typically labor to construct taxonomies from an unconstrained definition vocabulary, such as in Webster's Seventh. Other LDOCE work, principally at Cambridge University, has mainly concentrated on explicating the grammar codes for use with Unification based parsing (Alshawi, Boguraev and Briscoe 1985; Boguraev and Briscoe 1987; Boguraev 1987). Other efforts at extracting semantic information from machine-readable sources include recent work using LDOCE at Cambridge (Alshawi 1987), on locating meaningful patterns in definition text; searching for and explicating the so-called "defining formulas" of definitions using Webster's Seventh (Markowitz, Ahlswede and Evens, 1986); and recent work at IBM (Binot and Jensen 1987; Jensen and Binot 1987), that has taken an approach to interpreting definition parse trees, by applying a pattern matcher and a rule-based inference mechanism to assign MYCIN-like probabilities (Shortliffe 1976) to attachment alternatives (the numbers arrived at by intuition and tuning), again using Webster's Seventh as a knowledge base.

3.1. Relationship to Other CRL Approaches

The New Mexico State University Computing Research Laboratory (CRL), is partially funded by the New Mexico State Legislature, administered by its Science and Technology Commission as part of the Rio Grande Research Corridor. Of all the work at CRL towards an analysis of LDOCE (Wilks et al., 1987, 1988), this is perhaps the most direct. Rather than a statistical analysis intended to probabilistically determine the correct sense of a defining word according to its incidence of occurrence throughout the dictionary — this approach takes each sense definition as a largely independent unit for analysis. And instead of supplementing the dictionary material with other information, either by appeal to outside experts or through intuitively arrived at means — this approach limits itself to "what LDOCE has given," taking the dictionary more or less as found and trying to make the most of it.

The suppositions of this approach are —

1. that LDOCE *does* contain sufficient information to serve as a knowledge resource for text analysis, and
2. that that information can be processed in an effective way by programs.

The outside (external bootstrapping) information brought to bear is of the most general sort: a grammar for LDOCE sense definitions and a set of meaningful tree patterns.

We construe this to be a prototypical "book" approach. The global organization of the resulting lexicon mirrors the structure of LDOCE itself. That is, the hierarchies of grammar, type, into which LDOCE categorizes the world, as well as the specially restructured pragmatic hierarchy, are carefully preserved with this approach.

3.2. Psychological Plausibility

There is a thorny issue in the margin between Artificial Intelligence (AI) and Natural Language Processing (NLP) that relates to cognitive modelling and is perhaps best characterized as psychological plausibility (PP).

Certain identifiable camps in AI, principally those engaged in formal logic and theorem-proving, have no regard for PP notions. Other camps on the other extreme place PP into a central role, but these efforts are usually seen to be across the fuzzy border into the territory of Cognitive Science. Remaining in the "center," so to speak, is a large group that views PP as something either laudable or convenient. For this group, a feature or mechanism construed as psychologically plausible is a good thing, something to mention in publication, perhaps in a footnote; a lack of PP features or mechanisms is not noteworthy; but, somewhat curiously, a psychologically *implausible* feature or mechanism can be grounds for severe embarrassment and criticism. In the NLP wing of the AI center, for example, one is not "permitted" to write programs that process language with other than a single front-to-back pass (left-to-right in English), at least not without apology or justification. Under these criteria the context-setting mechanism of the lexicon-providing sub-system requires some justification (but, arguably, not apology).

The context-setting mechanism (section 2.1) is to construct a context structure for lexical disambiguation as a by-product of lexicon construction. In virtually every NLP system in existence, the first step in the eventual analysis of a text is retrieving the definitions of the relevant words and loading them into working memory -- this is mainly a time and space optimization, as there is normally no need or use for definitions of words not found in the text. This lexicon preprocessing performs a search-space reduction function analogous to the reduction achieved by loading a sub-language lexicon, such as is accessed by interactive systems in limited domains: the typical Natural Language Interface.

The lexicon-provider sub-system takes advantage of this pre-processing to activate the relevant portions of the restructured pragmatic hierarchy, and this hierarchy is ultimately used to influence word sense preference decisions. What this means is that the text has been scanned in advance of the actual language understanding process and it is possible to view this two-pass procedure as a psychologically implausible mechanism.

The first answer to this charge is in the form of a question, "What then, *would* constitute a psychologically plausible context-setting mechanism?" Of course, it is possible to argue that there is no such thing as context in language understanding, in which case this question is foolish, the point is moot, and the argument is lost. On the other hand, many would agree that context effects play a critical role in language comprehension, and then the question becomes something else entirely. For instance:

1. "What does it mean to "set context" (or, in another jargon, what is "priming")?"

or, alternatively,

2. "Who can say what gets "rolled into memory" and when or why?"

These last two questions might well be unanswerable. A man walks down the aisle in the QA section of a University library, reaches up and grabs a random book and opens it in the middle. What expectations, if any, will he have about what he is about to read? And, more central to this discussion, how are those expectations to be simulated by a computational artifact?

The fact is that the lexicon-provider sub-system described here is one of the few systems to address the question of context in language understanding in anything like a general way. The typical system either waits for one of a fixed set of key words or phrases and then retrieves a pre-formulated "script" structure, or else first constrains the lexicon (and hence the domain of discourse), and effectively sets context by eliminating the competing contexts from consideration at the outset, or both.

As a last line of defense against a charge of psychological implausibility it is possible to draw an analogy between the lexicon-provider context setting mechanism and the advise given to every freshman by every academic counseling service at every University on earth. For effective reading comprehension, so the conventional wisdom goes, first skim the text looking at titles, headings, and topic sentences of paragraphs, and then go back to the beginning and read more thoroughly word-for-word. If generously viewed in this light, the context-setting mechanism of the lexicon-provider is not psychologically implausible after all --- it is merely a heuristic mechanism for more effective reading comprehension.

4. Knowledge-Based Systems and the Future

The concerns of Natural Language Processing (NLP) and Knowledge Based Systems (KBS) research are usually only connected in the realm of Natural Language Interface (NLI). In KBS research the real problems are of acquiring the knowledge to put into the Knowledge Base (KB), maintaining consistency in the KB, and reasoning about the knowledge in the KB. Natural Language understanding generally figures in peripherally, if at all, in that it makes a nice way for the end-user to interact with the KBS, and a nice way for the human expert to create and modify the KB.

Recently, Nirenburg (1987) has wondered in print why so little effort has gone into studying the craft of the human language translator as an expert task. This does seem a natural area to explore and it is not difficult to draw analogies, at least, between expert problem solving as it is cast in KBS research and Natural Language understanding in the knowledge-based paradigm. In particular, both rely on a rich and structured source of knowledge and both face a major knowledge acquisition bottleneck on the road to implementing working systems.

It is not hard, then, to think of lexicon construction as a problem of knowledge acquisition since the lexicon serves in that relation to a language understanding system. Further, it is plausible to push the analogy to a second level, under the following assumptions:

1. that knowledge of language and knowledge of the world are both inseparably necessary for natural language understanding tasks;
2. that both sorts of knowledge exists in and can be extracted from text by programs; and
3. that dictionaries and encyclopedias, while being special purpose texts intended to encode knowledge of language and the world respectively, are both amenable to this kind of analysis with the intention of capturing these sorts of knowledge.

Central to this conception of automatic knowledge acquisition for knowledge-based systems is the notion of bootstrapping -- creating tools that produce some output, where that output is, in turn, fed through to other tools (or perhaps even the same tools), in order to produce yet other output which, after one or more iterations, proves suitable to some larger purpose. In this scenario something like the following occurs:

1. Dictionary entries are processed by means of a syntactic parser and pattern matcher to construct a lexical semantic knowledge-base (this much is the principle subject of this report);
2. A semantic parser is developed to process text, using the lexical semantic KB as a lexicon, and producing structured semantic objects to represent text elements (a system of this sort is in prototype development, see Slator (1988a, 1988c)); then
3. The semantic parser is set to work processing short encyclopedia entries, say, which results in a large collection of structured semantic objects encoding the world knowledge of the encyclopedia.

At the end of this bootstrapping cycle the semantic parser, now armed with both a lexical semantic KB as a lexicon, and a encyclopedia-derived KB of world knowledge, is closer to being ready to be applied to the more general problems of natural language understanding, such as text summarizing, question answering, and machine translation.

The obviously related work at this point is the CYC project at MCC Corporation (Lenat, Prakash, and Shepherd, 1986; Lenat and Feigenbaum, 1987) . This project has allotted ten years to encode a million entries from a desk encyclopedia, an estimated two person-centuries of work, into a large general purpose KB. Relatively little has been published on this project making it, as a consequence, difficult to critically evaluate. However, in a recent article (Leibowitz and Fjermedal, 1988), Lenat has claimed to have achieved "semantic convergence" where new concepts are being, at last, defined in terms of other previously encoded conceptual structures. The bootstrapping scheme outlined here is intended to parallel and, eventually, to compete with the CYC effort.

The final pay-off is that this notion of bootstrapping can be extended even a little farther. Once the contents of an encyclopedia have been captured in a structured knowledge base, the semantic parser could then also be applied to introductory technical texts on, say, blood diseases or geology, and could conceivably construct knowledge bases from these sources to contribute to the foundations of truly "expert" knowledge-based systems. However, no one can fail to notice, by this point, the highly speculative nature of these last paragraphs. We have now left the realm of describing computational artifacts and enter the realm of outlining future research programmes. The research described here is but the first steps towards these eventual goals.

References

ALSHAWI, HIYAN (1987). Processing Dictionary Definitions with Phrasal Pattern Hierarchies. *Computational Linguistics*, 13, 3-4, pp. 195-202. July-December.

ALSHAWI, HIYAN, BRANIMIR K. BOGURAEV, AND TED BRISCOE (1985). Towards a Dictionary Support Environment for Real Time Parsing. *Proceedings of the 2nd European Conference on Computational Linguistics*, pp. 171-178. Geneva.

AMSLER, ROBERT A. (1980). The Structure of the Merriam-Webster Pocket Dictionary. *Technical Report*. (TR-164). University of Texas at Austin. Ph.D. Thesis.

AMSLER, ROBERT A. (1981). A Taxonomy for English Nouns and Verbs. *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, pp. 133-138. Stanford.

AMSLER, ROBERT A. (1986). Deriving Lexical Knowledge-Base Entries from Existing Machine-Readable Information Sources. *Unpublished.*

AMSLER, ROBERT A. AND JOHN S. WHITE (1979). Development of a Computational Methodology for Deriving Natural Language Semantic Structures via Analysis of Machine-readable Dictionaries. *NSF Technical Report*. (MCS77-01315).

BARR, AVRON AND EDWARD A. FEIGENBAUM (EDS.) (1981). *The Handbook of Artificial Intelligence, Vol. 2*. Los Altos, CA: W. Kaufman. Q335/H36.

BINOT, JEAN-LOUIS AND KAREN JENSEN (1987). A Semantic Expert Using an Online Standard Dictionary. *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 2, pp. 709-714. Milan, Italy: IJCAI-87. To appear as an IBM Research Report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

BOGURAEV, BRANIMIR K. (1979). Automatic Resolution of Linguistic Ambiguities. *University of Cambridge Computer Laboratory Technical Report*. (No.11). Cambridge, UK: University of Cambridge Computer Laboratory.

BOGURAEV, BRANIMIR K. AND TED BRISCOE (1987). Large Lexicons for Natural Language Processing: Exploring the Grammar Coding System of LDOCE. *Computational Linguistics*, 13, 3-4, pp. 203-218. July-December.

CARTER, DAVID M. (1984). An Approach to General Machine Translation Based on Preference Semantics and Local Focussing. *Proceedings of the 6th European Conference on AI (ECAI-84)*, pp. 231-238. Pisa, Italy.

CARTER, DAVID M. (1987). *Interpreting Anaphors in Natural Language Texts*. Chichester, UK: Ellis Horwood.

CHODOROW, MARTIN S., ROY J. BYRD, AND GEORGE E. HEIDORN (1985). Extracting Semantic Hierarchies from a Large On-Line Dictionary. *Proceedings of the 23rd Annual Meeting of the ACL*, pp. 299-304. Chicago, IL.

FASS, DAN C. (1986). Collative Semantics: An Approach to Coherence. *Computing Research Laboratory Memorandum*. (MCCS-86-56). Las Cruces, NM: New Mexico State University.

FASS, DAN C. (1987). Semantic Relations, Metonymy, and Lexical Ambiguity Resolution : A Coherence-Based Account. *Proceedings of the 9th Annual Cognitive Science Society Conference*, pp. 575-586. Seattle, WA: University of Washington.

FASS, DAN C. (1988). An Account of Coherence, Semantic Relations, Metonymy, and Lexical Ambiguity Resolution. In *Lexical Ambiguity Resolution in the Comprehension of Human Language*. Edited by Steve L. Small, Gary W. Cottrell, and Michael K. Tanenhaus. pp. 151-178. Los Altos, CA: Morgan Kaufmann.

HUANG, XIUMING (1984). The Generation of Chinese Sentences from the Semantic Representations of English Sentences. *Proceedings of International Conference on Machine Translation*. Cranfield, UK.

HUANG, XIUMING (1988). XTRA: The Design and Implementation of A Fully Automatic Machine Translation System. *Computing Research Laboratory Memorandum*. (MCCS-88-121). Las Cruces, NM: New Mexico State University.

JENSEN, KAREN AND JEAN-LOUIS BINOT (1987). Disambiguating Prepositional Phrase Attachments by Using On-Line Dictionary Definitions. *Computational Linguistics*, 13, 3-4, pp. 251-260. July-December.

LEIBOWITZ, MICHAEL AND GRANT FJERMEDAL (1988). Brain Trusts. *Omni Magazine*, 10, 9, pp. 44-50. June.

LENAT, DOUGLAS B. AND EDWARD A. FEIGENBAUM (1987). On the Thresholds of Knowledge. *Proceedings of the 10th International Joint Conference on Artificial Intelligence*. Milan, Italy: IJCAI-87.

LENAT, DOUGLAS, B., MAYANK PRAKASH, AND MARY SHEPHERD (1986). CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. *AI Magazine*, 7, 4, pp. 65-85.

MARKOWITZ, JUDITH, THOMAS AHLSWEDE, AND MARTHA EVENS (1986). Semantically Significant Patterns in Dictionary Definitions. *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pp. 112-119. New York.

MASTERMAN, M. (1957). The Thesaurus in Syntax and Semantics. *Mechanical Translation*, 4, 1-2.

MILLER, GEORGE A. (1985). Dictionaries of the Mind. *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, pp. 305-314. Chicago.

NIRENBURG, SERGEI (1987). Knowledge and Choices in Machine Translation. In *Machine Translation: theoretical and methodolocal issues*. Edited by Sergei Nirenburg. pp. 1-21. Cambridge, UK: Cambridge University Press.

PROCTER, PAUL ET AL. (1978). *Longman Dictionary of Contemporary English (LDOCE)*. Harlow, Essex, UK: Longman Group Limited.

PULMAN, STEPHEN G. (1983). Generalized phrase structure grammar, Earley's algorithm, and the minimisation of recursion. In *Automatic Natural Language Parsing*. Edited by Karen Sparck-Jones and Yorick A. Wilks. pp. 117-131. Chichester/New York: Ellis Horwood/Wiley.

QUIRK, RANDOLPH, SIDNEY GREENBAUM, GEOFFREY LEECH, AND JAN SVARTIK (1972). *A Grammar of Contemporary English*. Harlow and London, England: Longman Group Limited.

QUIRK, RANDOLPH, SIDNEY GREENBAUM, GEOFFREY LEECH, AND JAN SVARTIK (1985). *A Comprehensive Grammar of English*. Harlow and London, UK: Longman Group Limited.

SHORTLIFFE, EDWARD H. (1976). *Computer-Based Medical Consultation: MYCIN*. Elsevier Science Publishers B.V., North-Holland.

SLATOR, BRIAN M. (1988). Lexical Semantics and a Preference Semantics Parser. *Memoranda in Computer and Cognitive Science*. (MCCS-88-116). Las Cruces, NM: Computing Research Laboratory, New Mexico State University.

SLATOR, BRIAN M. (1988). PREMO: the PREference Machine Organization. *Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 258-265. Denver, CO. June, 13-15.

SLATOR, BRIAN M. (1988). Constructing Contextually Organized Lexical Semantic Knowledge-Bases. *Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 142-148. Denver, CO. June, 13-15.

SLATOR, BRIAN M. AND YORICK A. WILKS (1987). Towards Semantic Structures from Dictionary Entries. *Proceedings of the Second Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 85-96. Boulder, CO. Also as CRL Memo MCCS-87-96.

SLOCUM, JONATHAN (1985). Parser Construction Techniques: A Tutorial. *The 23rd Annual Meeting of the Association for Computational Linguistics*. Chicago, IL.

SPARCK-JONES, KAREN (1964). Synonymy and Semantic Classification. *Ph.D. Thesis*. Cambridge, UK: University of Cambridge. Published in Edinburgh Information Technology Series (EDITS), Sidney Michaelson and Yorick A. Wilks (Eds.), Edinburgh University Press: Edinburgh, Scotland, 1986.

WALKER, DONALD E. AND ROBERT A. AMSLER (1986). The Use of Machine-Readable Dictionaries in Sublanguage Analysis. In *Analyzing Language in Restricted Domains*. Edited by Ralph Grishman and Richard Kittredge. pp. 69-84. Hillsdale, NJ: Lawrence Erlbaum.

WILKS, YORICK A. (1972). *Grammar, Meaning, and the Machine Analysis of Language*. London: Routledge and Kegan Paul.

WILKS, YORICK A. (1975). An intelligent analyzer and understander of English. *Communications of the ACM*, 18, 5, pp. 264-274. Reprinted in "Readings in Natural Language Processing," Edited by Barbara J. Grosz, Karen Sparck-Jones and Bonnie Lynn Webber, Los Altos: Morgan Kaufmann, 1986, pp. 193-203.

WILKS, YORICK A. (1975). A Preferential Pattern-Seeking Semantics for Natural Language Inference. *Artificial Intelligence*, 6, pp. 53-74.

WILKS, YORICK A. (1978). Making Preferences More Active. *Artificial Intelligence*, 11, pp. 75-97.

WILKS, YORICK A., DAN C. FASS, CHENG-MING GUO, JAMES E. MCDONALD, TONY PLATE, AND BRIAN M. SLATOR (1987). A Tractable Machine Dictionary as a Resource for Computational Semantics. *Proceedings of the Workshop on Natural Language Technology Planning*, Sept. 20-23. Blue Mountain Lake, NY. Also as CRL Memo MCCS-87-105. To appear in "Computational Lexicography for Natural Language Processing." Branimir K. Boguraev and Ted Briscoe (Eds.). Harlow, Essex, UK: Longman. 1988

WILKS, YORICK A., DAN C. FASS, CHENG-MING GUO, JAMES E. MCDONALD, TONY PLATE, AND BRIAN M. SLATOR (Forthcoming - 1988). Machine Tractable Dictionaries as Tools and Resources for Natural Language Processing. *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88)*. Budapest, Hungary. Aug. 22-27.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Distributed Causal Reasoning for Knowledge Acquisition: A Functional Approach to Device Understanding*

Jon Sticklen
AI/KBS Group - CPS Dept
A 714 Wells Hall
Michigan State University
East Lansing, MI, USA 48824
sticklen@cpswh.cps.msu.edu

B. Chandrasekaran
LAIR - CIS Dept
2036 Neil Ave Mall
Ohio State University
Columbus, OH, USA 43210
chandra%osu-20@tut.cis.ohio-state.edu

W.E. Bond
McDonnell Douglas
Research Lab
St. Louis, MO, USA 63166
c0567@umrvmb.bitnet

Abstract

We describe an approach for understanding devices which provides leverage for the knowledge acquisition task. In our *functional approach*, knowledge is acquired in three steps: the device is decomposed to an ensemble of subdevices, the functions/goals/purposes of each subdevice are stated abstractly, and finally, the way(s) each function/goal/purpose is accomplished is represented. The heart of our approach is a distribution of causal knowledge into fragments which are indexed by the functions/purposes/goals of the device. We show that the functional approach supports automated knowledge acquisition for diagnostic problem solving, and we argue that it also supports manual knowledge acquisition for device knowledge.

1. INTRODUCTION

Efficient knowledge acquisition (KA) is a lynchpin for practical application of Knowledge Based Systems. The realization of this fact has spawned two lines of research within the AI community. Along one path, research has aimed at providing general representation facilities and a programming environment supportive of KA activities; for example [Abrett et al 1986]. The second research direction has centered on exploiting strengths for undertaking KA inherent in *particular* problem solving architectures; for example [Eshelman et al 1986].

Recently, Sticklen and Chandrasekaran completed a research project, MDX2, which from a KA perspective is best described as following the second of these research paths [Sticklen 1987]. MDX2 is an operational diagnostic system in mixed domains of clinical medicine. It was designed and constructed utilizing the broad tenants of the Generic Task Approach to Knowledge Based Systems [Chandrasekaran 1983; [Chandrasekaran 1985; Chandrasekaran 1986]. The distributed problem solving architecture of MDX2

* Partial support for this research is from the McDonnell Douglas Independent Research and Development program. Dr. Sticklen is also currently supported as an Ameritech Fellow at MSU, and by equipment grants from Apple Computer and Texas Instruments. Part of the work reported here was completed at the Laboratory for AI Research, Ohio State University, under NLM Training Grant T15 LM 07023, NLM Grant R01 LM 04298, and NSF Grant MCS 8305032.

includes both highly compiled modules, and a deeper level problem solving unit. In the current report, we will focus on issues of deep level representation and reasoning about devices, and on one way in which a deep level problem solver can interact with a more compiled problem solver.

One basis for interest in deep level problem solving stems from problems of KA. Suppose that a new device is engineered to perform a particular task. One way to construct a trouble shooting, knowledge based system for this device would be to acquire the knowledge necessary to perform the desired diagnostic task from experienced maintenance people. But if the device is new, then such experiential knowledge would likely not be totally available. Moreover, even if such knowledge is available, it would be expensive to acquire. On the other hand if it were possible to make use of the design knowledge used to construct the device as a basis for a troubleshooter, then the KA bottleneck would be broken since this knowledge would be available immediately.

As well as the issues of KA, there are other considerations that have led to calls for the development of deep reasoning capabilities in Knowledge Based Systems [Hart 1982; Michie 1982]. Although there is general agreement that deep reasoning capabilities are desirable there is no apparent consensus on how to model such deep knowledge, nor on how to reason about it. In fact, there is no clear consensus on what the terms "deep level understanding" or "deep level reasoning" mean.

In this report, we describe one approach to modeling and reasoning about deep knowledge of devices: the *functional approach*. We first discuss perspectives from which to view knowledge acquisition activities. Then we describe our functional approach to modeling deep knowledge and to reasoning on the basis of the model, including how a functional reasoner can performed a well defined type of automated knowledge acquisition. Next we present an actual case from the MDX2 project over our implemented representation of a part of human body physiology. Finally, we discuss points of contact with other research, and current directions which our research is taking.

2. DIMENSIONS FOR KA

Before beginning to discuss our deep reasoning method, it is important to develop a clear concept of the KA goals we would hope to achieve with any reasoning methodology, deep or otherwise. Bylander and Chandrasekaran have argued that knowledge acquisition should not be considered in isolation, but rather should be conceived of as an adjunct to some specified problem solving task [Bylander and Chandrasekaran 1986]. An extrapolation from their argument is that considering KA in isolation is "putting the cart before the horse" in the following sense. Before one can specify how to acquire knowledge, one must specify the type of knowledge one needs, and that in turn is largely set by the problem solving task one wants to accomplish.

There are two orthogonal dimensions along which problem solving techniques can be judged for strength of support for KA activities:

1. Ability to support manual knowledge acquisition.

   An architecture for problem solving comes with some representational scheme for expressing the domain knowledge which will be utilized. To the extent that this representational scheme includes principles of knowledge organization and epistemic primitives mirroring those of the domain expert, the KA task will be eased.

   Although this notion is easy enough to enunciate, documenting strengths for KA for any particular problem solving methodology along this dimension will require extensive psychological testing. To date, such psychological testing has by and large not been undertaken; arguments have typically been based on observed ease of domain knowledge elucidation without rigorous testing to support the claims.

2. Ability to support automated knowledge acquisition.

   Supposing that one can efficiently elicit the domain knowledge necessary

for the functioning of a particular type of problem solver, another round of KA is possible. If the problem solver generates knowledge in a form useful for another problem solver, then automated KA has occurred. Call the initial problem solver PS-Deeper, and the one that will use the generated knowledge PS-LessDeep.

Note that there are in general two ways in which PS-Deeper and PS-LessDeep can interact. PS-Deeper could be used to derive all the knowledge PS-LessDeep requires before PS-LessDeep is called upon to solve some problem. A second valid interaction method is for PS-LessDeep to call on PS-Deeper *during* problem solving with a request to provide a particular types of knowledge important for the current problem being solved.

In order to claim that a particular problem solving architecture has strength along this dimension of support for KA, one needs to discuss the problem solving architecture (PS-Deeper), and how it generates knowledge useful for the second problem solver (PS-LessDeep).

One problem solving methodology can have strength along both of these dimensions. In fact, we will claim in the end that the Functional Approach to device understanding, as we describe it below, has strength both for manual knowledge acquisition of understanding about devices, and for "compiling" knowledge useful for a diagnostic problem solver. Support for the claim of strength for manual knowledge acquisition will not be based on hard psychological experiments, but rather on the reported ease of use of our techniques. Support for the claim of strength for the Functional Approach will be based on the exposition in the bulk of this paper of our method, and the way that a "functionally based" problem solver can interact with a diagnostic system.

3. THE FUNCTIONAL APPROACH

Prior research in deep reasoning methods has typically fallen into two broad areas: the causal net approach [Patil 1981; Pople 1977; Pople 1979; [Weiss et al 1977] and the naive physics approach [Bylander 1986; deKleer and Brown 1984; Forbus 1984; [Kuipers 1984; Kuipers and Kassirer 1984]. The causal net approach centers on representing deep knowledge via a causal net structure; deep reasoning then means controlling the process of navigating the causal net. The naive physics approach centers on *deriving* the functionality of a device from the function and structure of its constituent subdevices, based on a view of causality that largely centers on temporal precedence.

Although the two approaches; causal net and naive physics; are quite different, they both begin with the intuition that human reasoning about devices typically includes some aspect of causality. In the causal net approach, the causality is directly woven into a comprehensive network representing state descriptions of the device being modeled. In the naive physics approach, causality of device behavior in the small is used to compute state changes in the large; i.e., changes that characterize the entire device.

The functional approach (FA) begins with the same intuition: causality *is* important in deep level reasoning about devices. From that beginning however, the FA proposes an intermediate position between the reliance on deriving all large scale cause-effect relations of naive physics, and the monolithic representation of all cause-effect relations of the causal net approach. We will return to discuss the relationships between the causal net approach, the naive physics approach, and the functional approach to device understanding towards the end of this paper.

3.1. EXPLICIT STARTING POINTS

Our approach springs from the broad framework of the Generic Task theory of Knowledge Based Systems [Chandrasekaran 1983; Chandrasekaran 1985; [Chandrasekaran 1986], extends the earlier framework of Sembugamoorthy and Chandrasekaran [Sembugamoorthy and Chandrasekaran 1986], and builds on our initial conception of how a compiled level problem solver and a deep level problem can interact [Sticklen and Chandrasekaran 1985].

There are a number of intuitions that comprise the starting point for development of the functional approach; we first list these intuitions.

3.1.1. Intuition 1: Limitation to Devices

First, we limit our concern to the world of devices. We define a device as "a naturally occurring or artifactual assemblage whose purposes/goals/functions are known." We limit consideration to devices because we will make use of the functionality the physical device has.

Any physical object can in principle be described in terms of its physical properties, but once we understand how we may *use* a device, we understand it at a different level. This device level is distinguishable from the "attribute description" level by the *indexing capability* we gain at the device level.

3.1.2. Intuition 2: Device Decomposition

As we consider a complicated device we manage its complexity by recursively *decomposing* it into a number of components until, as the decomposition continues, we eventually reach a level at which we can grasp how that level of component operates. Understanding the operation of the overall device can then be stated in terms of the operation of the components.

3.1.3. Intuition 3: Organization of Understanding Around Functions:

As device decomposition proceeds, we organize our understanding of the device in terms of the purposes to which the device will be put; i.e. in terms of the functions of the device. Most importantly, we understand [1] functionality of the overall device in terms of the functionality of its components.

3.1.4. Intuition 4: Composability of Understanding:

Intuition #2 above dealt with decomposing a complex device to a number of sub-devices. Intuition #3 is that we organize our understanding about each of those sub-devices in terms of the functions/goals of each. Both #2 and #3 can be characterized as dealing with the *static* representation of device understanding. Since, however, the core idea is that we break through the complexity of device understanding by a process of decomposition, we must be able to bring together relevant fragments from the totality of our understanding of a particular device in response to a particular situation in which the device finds itself. Thus, as a starting point in our treatment of complex device understanding, we include the notion that the model of device understanding and reasoning over the model must allow a *dynamic* process of composition in response to a given situation.

3.1.5. Information Processing Task at the Functional Level

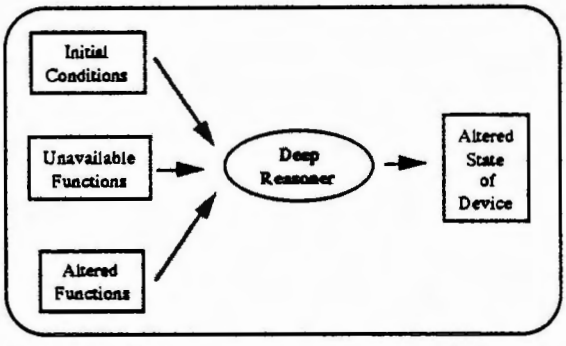


Figure 1: Information Processing Task for our Functional Level Understander

The Information Processing Task (IPT) [2] we set for functional level reasoning is a focused consequence finding as shown in Figure 1. Device representation, include all normal (i.e., default) device functions and state variable values are include in the "Deep Reasoner" itself. Three types of input are specified:

1. Any non-default values for state variables of the device.
2. A list of any device functions which should be considered as "broken", i.e. unavailable.
3. A list of any device functions which have been altered from "normal functionality", along with a representation of how the abnormal function operates.

The output for the IPT is a list of the consequent changes in the state variables of the device which

1. Note we say understand - *not* derive.

2. The idea of the IPT was first discussed by Marr [Marr 1982].

result given the input. Thus, from the highest level, the computational purpose which we set for function level reasoning can be characterized as a type of *consequence finding*.

In the remainder of this section, we will describe our methods for representing and reasoning about devices. We will center the discussion in large part on a trivial example of an ordinary clothespin. Having laid the foundation for understanding here, in the following section we will go through a more realistic example to demonstrate the power of the functional approach in more detail.

3.2. Functional Representation and Reasoning

3.2.1. Representation

The functional level description of a device includes three core components: a description of device structure, a description of the functions of the device, and a description of the behaviors of the device which achieve the abstractly stated functions. To illustrate the various components of the functional representation, consider the simple device shown in Figure 2 and the partial functional description shown in Figure 3 and Figure 4.

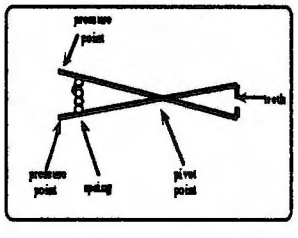


Figure 2: Simple Device - A Clothespin

The device description of Figure 3 includes two items: a listing of the sub-devices out of which the current device is composed and a listing of the functionalities of which the device is capable. Note that by simply looking at Figure 2, it is possible to conceive of a clothespin as made of many different sets of sub-devices. A characteristic of the functional level description of a device is that of the many possible ways of "cutting up the pie" of a device, one way has been selected around which reasoning will be centered.

```
Device : CLOTHESPIN
  SubDevices: (ARM,
               PIVOT,
               SPRING,_ )
  Functions: (Open,
              Close,
              Hold,_ )
```

Figure 3: Part of Device Specification for Clothespin

Function:	***Open* of *CLOTHESPIN***
ToMake:	teeth-more-open
Provided:	(pressure-point-force-applied *GreaterThan* restoring-force)
By:	*OpenBehavior*

Figure 4: Open Function of Device Clothespin

The function description of Figure 4 contains three items:

1. a *Provided* clause which states the conditions under which the function is applicable. The Provided clause may be thought of as a pre-condition for the behaviors which carry out the function.
2. a *ToMake* clause which states the result which will be obtained if the invoked function completes successfully. The ToMake clause may be thought of as a post-condition for the behaviors that carry out the function.
3. a *By* clause which points to the behavior(s) carrying out the function.

The function declarations within the functional description provide a means of knowing what can be achieved (the ToMake clause), what must be true for a given function to be applicable (the Provided clause), and a pointer for where to look if a more detailed description is needed of how the function achieves its goal (the By clause).

The description of behavior at the functional level contains two ingredients, as illustrated in Figure 5.

1. Partial state descriptions of the device. These descriptions take two forms, as illustrated in Figure 5: the initial "box" in a

behavior is a predicate test on some state variable of the device; the other "boxes" of a behavior are statements about changes to state variables of a the device.

2. Annotated links. These links represent both the logical flow of one state leading to another, and the *reason* that one state follows another *via* the links. Note that the annotation *could* be a pointer to another function or behavior, and hence could be decomposed if necessary, or it could be to a fragment of world knowledge which could not be further decomposed.

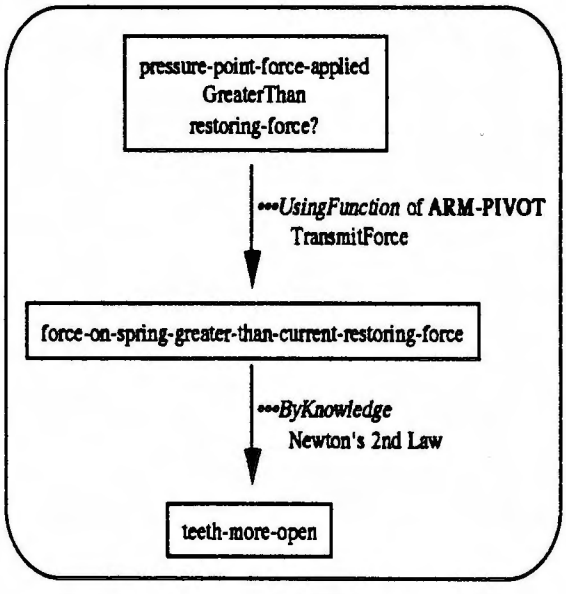


Figure 5: The Implementing Behavior for the Open Function of the Device Clothespin

From the above discussion, the following facets of the functional description of a device emerge. First, the functional representation of a device is a conceptual abstraction of what a device *is* and how it *works*. The "what it is" part is represented as a collection of sub-devices related by a *ComponentOf* relation. The "how it works" is represented as the functionalities of which it is capable and the behaviors that accomplish those functions.

Second, the functional description allows a natural modularity in understanding devices. This modularity is important in three specific ways. First, a sub-device of the overall device may be replaced with another totally different sub-device which accomplishes the same functions.

Third, in understanding from the top level how the device functions, we are normally led via a chain of

device → function → behavior

→ sub-device •••

to lower and lower levels of sub-devices. However, this path of understanding may be terminated before the lowest levels of the device are reached. Once a level is reached at which a particular functionality of some underlying sub-device may be "assumed true", then further probing along the current path is unnecessary. This ability to probe only as far as needed follows directly from the modularity of representation adopted. Put another way, in the functional approach to device understanding, there is a implicit natural "layering of understanding" from the most abstract levels of device description to the most detailed.

Finally, and related to the last point, the functional approach to device representation includes as a central pillar the temporal flow variant of causality. Although there is no unitary causal net in a functional representation, each behavior can be thought of as a causal net fragment. We return to this point in the discussion of Section 7 below.

3.2.2. Function-Based Consequence Finding

As we noted above, the Information Processing Task we undertake is a type of consequence finding. Starting from the approach to device representation described above, it is straightforward to produce consequence finding results. The consequence finding is undertaken in response to a particular set of conditions, and amounts to building up a full state change diagram from the fragments that exist in the behaviors of the functional representation. In other words, we will build a specialized causal net for the particular circumstance under consideration.

In detail, the consequence finding algorithm is as follows.

1. Specify the initial conditions as illustrated in Figure 1. The variables of the device

have associated with them "default values" so that only device variable bindings which are not the normal state of the device need be set. Likewise, only missing functions, or altered functions need to be explicitly input.

2. Determine the functions/behaviors that should be used as a starting point. Once the initial conditions of the device are specified, then it becomes possible to use those initial conditions to *index* behaviors and functions of the device that would be applicable under those starting conditions. After a round of "filtering" to remove redundant functions/behaviors[3], we are left with a set of functions/behaviors we will call the "invocable functions/behavior". If there are no invocable functions/behaviors the functional reasoner halts.

3. Starting with the invocable functions/behaviors from the previous step, construct a new state-change graph structure *particularized* for the current situation. Call this new structure a *Particularized State Diagram* (PSD). Each node in the PSD will be a partial state description in the same sense as before - that is, as a pointer to a variable of the device and as a statement about how that variable is altered.

   The PSD is constructed by traversing each of the starting behaviors.

   a. When at a partial state, place into the PSD which is being built a corresponding node to mark a partial state change, and, in addition, update an associated state variable database accordingly.

   b. If we are at an annotation which is non-decomposable then remember that succeeding partial states assume whatever the annotation points to but make no changes in the PSD that is being built.

   c. If we are at an annotation which is decomposable (that is, another function or behavior), remember that succeeding partial states assume the function/behavior pointed to (as in b) and also at this point in the construction expand the function/behavior pointed to whether possible. To determine whether a given function/behavior may be expanded, check its starting predicates.

4. The process of expanding the annotation links continues until there remain no more links that are decomposable.

Put simply, the PSD is built by following all decomposable annotations that were in the starting behaviors and expanding them recursively until what is left is a PSD which includes only partial state transitions. Each node in the PSD contains knowledge of the state variable it alters and the nature of the alteration. In addition, each node contains a listing of the assumptions under which this state change takes place. Once the PSD has been constructed, it is easy to determine what the effect on the device will be by traversing the PSD and noting cumulative changes that take place in the state description variables of the device.

4. USING A FUNCTIONAL REASONER TO ACQUIRE DIAGNOSTIC KNOWLEDGE

In Section 2, we argued that one type of knowledge acquisition strength which a problem solving architecture can have is the ability to generate knowledge which is useful for another problem solver. In previous research, we developed a comprehensive architecture for diagnostic problem solving, the MDX2 architecture [Sticklen 1987]. Although not limited to one type of problem solving, the core of MDX2 at the compiled level is classification based. In this section, we describe how a classification problem solver can direct the activities of a functionally-based consequence finder, then use the results from the function level as newly acquired diagnostic knowledge.

As a preface to this discussion, the rudiments of hierarchical classification should be understood.[4]

3. An example of the redundancy we need to "filter" would be two functions, call then F_a and F_b, such that both have the same *Provided* clause (i.e., same precondition), and such that F_a contains a knowledge pointer to the function F_b. Then we want to filter out F_b.

For current purposes, it is enough to know that hierarchical classification is undertaken in a hierarchically organized hypothesis space, in which each of the nodes of the hierarchical organization represents an individual hypothesis. Problem solving in this hierarchical organization proceeds top down following a control regime called "establish-refine" - a primitive hypothesis first must establish itself; i.e., determine if it is a viable hypothesis for the problem solving case at hand; then if it is established, it refines itself; i.e., it gives control to its subordinate hypotheses in the hierarchy asking them to establish then refine. Control passes from high, more general categories of description, to deep, more specific categories. For example, in the classification hierarchy for diagnosing infectious disease, of Figure 6, would be entered at the top of the hierarchy, InfectiousDisease. If that most general category could be established, then the problem solver would go on to consider the types of infectious disease, BacterialDisease, ViralDisease, ...

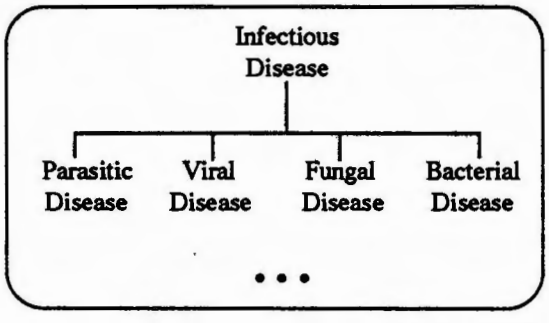


Figure 6: High Level Part of Infectious Disease Hierarchy

For this discussion, it is important to note that a critical part of the operation of each classification specialist is to be able to establish itself. At the compiled problem solving level, the general method of undertaking establishment is to prestore with each hypothesis patterns which should be looked for to provide evidence for (or against) the plausibility of the hypothesis. (A discussion of this type of "experience based matching" can be found in [Sticklen et al 1987].) But when such patterns are prestored, the assumptions under which the patterns are valid are usually not represented. This can lead to situations in which, for example, a bacterial infection may present in a very atypical pattern, because one of more of the assumptions leading to patterns of patient signs and symptoms about infectious disease are not met. It still may be suspected that bacterial disease is present, yet the plausibility of bacterial disease may be very low.

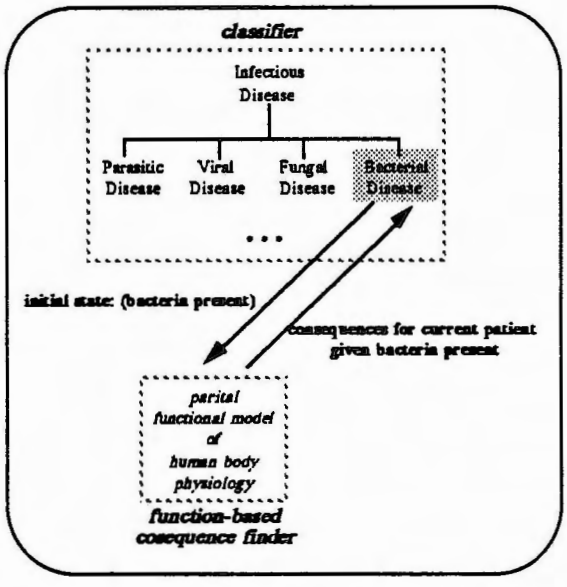


Figure 7: Classifier Acquiring Knowledge by Interacting with Function Level Problem Solver

Suppose there is available a functional representation of relevant parts of human body physiology. Then if the compiled level hypothesis of BacterialDisease carries with it the knowledge that "BacterialInfection means the presence of foreign bacteria.", then when bacterial disease is suspected, but the plausibility of bacterial disease seems to be low, it would be possible to invoke the functional reasoner with the starting condition "bacteria present", derive the consequences for the current patient of this starting condition, and use those consequences as new patterns with which to try to establish the hypothesis BacterialDisease. This overall architecture for interaction between a compiled level classification problem solver, and a deeper level functionally-based problem solver is depicted in Figure 7. In the next section, more detail of this interaction will be seen as we discuss an actual problem solving case.

4. For more in depth treatments of this subject, see [Chandrasekaran 1982; Chandrasekaran and Mittal 1983; [Bylander and Mittal 1985].

5. EXAMPLE - THE HUMAN COMPLEMENT SYSTEM

So far in our exposition of functional approach we have described the representational and reasoning methodology which we follow, and shown how a compiled level classification problem solver can use a functional consequence finder to acquire new knowledge. We now illustrate all these points by describing how we represent and reason about a part of the human body's immune system: the complement subsystem. We first describe the relevant physiology at a simplified level, then go on to take a vertical slice through our representation of the complement system to demonstrate the methods of functional representation and reasoning.

5.1. DOMAIN OF THE EXAMPLE: THE SIMPLIFIED COMPLEMENT SYSTEM

The Complement System of the human body is a sub-system of the Host Defense System. The duties of the Host Defense System center on protecting the body from foreign material - either animate (e.g., bacteria) or inanimate (e.g., pollen grains). The various sub-systems of the Host Defense System are redundant over two top level functions: distinguishing between "self" and "non-self" material, and disposing of "non-self" material.

In an evolutionary sense, the Complement System is one of the oldest sub-systems of the Host Defense System. From a purely biochemical viewpoint, the Complement System consists of a number of chemicals which are usually found in the blood. These chemicals are called the Complement Factors, and are numbered from 1 to 9 - Complement Factor 1 (C1) to Complement Factor 9 (C9). The Complement System is not capable of sensing and responding to all types of non-body material. One of the types of foreign material for which it is particularly tuned is a broad range of bacterial invaders.

The biochemical task which the C1-C9 molecules undertake, in conjunction with other enzymatic constituents of the blood, is to assemble a molecule capable of directly killing bacterial invaders, via a process called *lysis*; hence, the molecule is known as lytic material. An organism need not continuously produce lytic material, but rather should produce it as needed. Not only would the assembly of lytic material in the absence of bacteria be wasteful, it would also be dangerous since the lytic material can lyse normal body cells under some conditions.

The overall job of the Complement System consists of two parts: recognize that lytic material is required by noting the presence of bacteria, and assemble the lytic material. From a functional viewpoint, the Complement System is a superb example of a well-designed, redundant system. There are three major subsystems in the Complement System, as summarized in Figure 8.

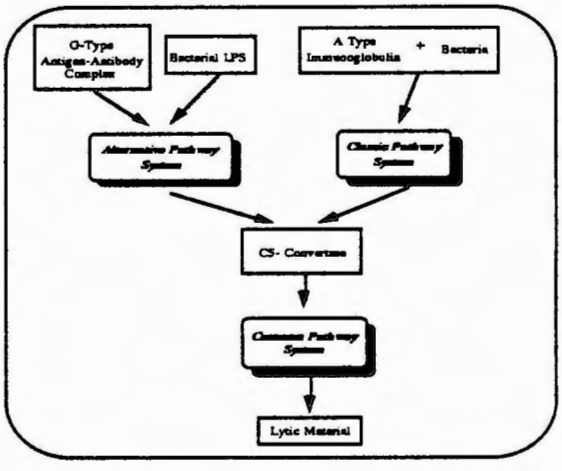


Figure 8: High Level View of the Complement System

1. *The Classic Pathway System.* This sub-system is one of the two recognition oriented sub-systems. It recognizes the presence of bacteria by direct recognition of molecular complexes activated in other sub-systems of the Host Defense system in response to bacterial contact. Once activated, the Classic Pathway system results in the production of a molecule called "C5-Convertase", the activating molecule for the Common Pathway System.
2. *The Alternative Pathway System.* This is the second of the two recognition-oriented systems. The Alternative Pathway System detects the presence of bacteria in two ways: direct recognition of bacteria, and

recognition of a by-product of bacterial metabolism which is called bacterial-LPS. Once either one of both modes of recognition are accomplished, the Alternative Pathway System will, like the Classic Pathway, produce C5-Convertase.

3. *The Common Pathway sub-system.* The purpose of this sub-system is to produce the lytic material which can directly attack bacteria. Once C5-Convertase is present, the Common Pathway System will assemble, through a series of steps, a complex molecule with the ability to lyse bacteria.

The Complement System is only a small part of the overall Host Defense System, which in turn is but a small functional part of the human body. Nonetheless, the Complement System offers substantial complexity in and of itself. In addition, the Complement System crystallizes several representational challenges. One of the most interesting is that in order to represent the complement system, we must deal with abstract sub-systems. By abstract, we mean sub-systems that could not be identified by doing a structural decomposition. Many of the biological systems of the human body have this characteristic.

5.2. Representing A Vertical Slice of the Complement System

In this section we will lay out a part of our functional representation for the Complement System. In the exposition to follow we focus heavily on the representation of behavior. Further, we discuss only a portion of the total representation following from one particular high level starting point; i.e., we will look at a vertical slice through the representation by following how one behavior can lead us to another deeper one.

Before we set out on following that vertical slice, however, let us first get a top level view of the overall representation for the Complement System. Figure 9 shows the component decomposition for the entire system; Figure 10 shows the functions which each of the components is capable of, and the behaviors which implement these functions.

Let us take as a starting point the presence of the metabolic by-product of bacterial metabolism, bacterial-LPS. From the above discussion, it should be remembered that bacterial-LPS is a trigger of the Alternative Pathway System and that the chain of events beginning with the presence of bacterial-LPS will ultimately lead to the production of lytic material capable of destroying the bacteria which produce the bacterial-LPS.

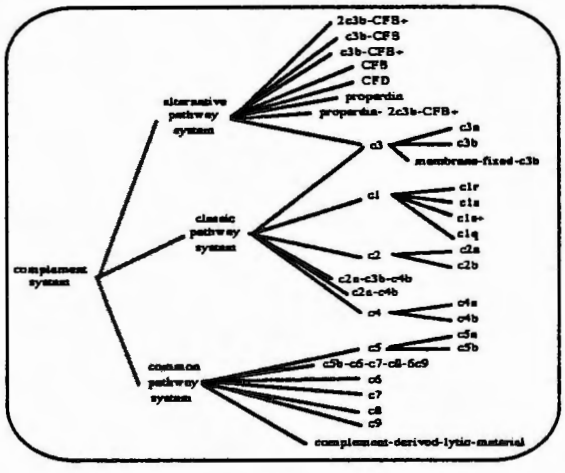


Figure 9: Component Decomposition for the Complement System

Figure 11 shows a top level behavior of the Complement System: in fact, the behavior that responds to the presence of bacterial-LPS. This behavior, the "make-anaphylotoxin-lytic-material-LPS-activated-alternative-pathway" behavior, is initiated when there is bacterial-LPS present, as indicated by the starting state variable predicate in the behavior. Following that, C5-convertase will be produced. The way it will be produced can be understood by following the link annotation shown to the behavior "make-anaphylotoxin-opsonin-LPS-activated-alternative-path" - which we will do in a moment. At this point, let us continue to follow the top level behavior.

Following the production of C5-convertase, complement-derived-lytic-material will be produced and, in addition, anaphylotoxin will be produced. Anaphylotoxin is a material which has no use in the limited context of the Complement System except to indicate that a certain functionality is being achieved. Beyond the Complement System however, anaphylotoxin is a powerful instigator of a reaction known as the Inflammatory Response, which manifests itself by localized soreness and redness due largely to a dilation of local blood vessels. The connection

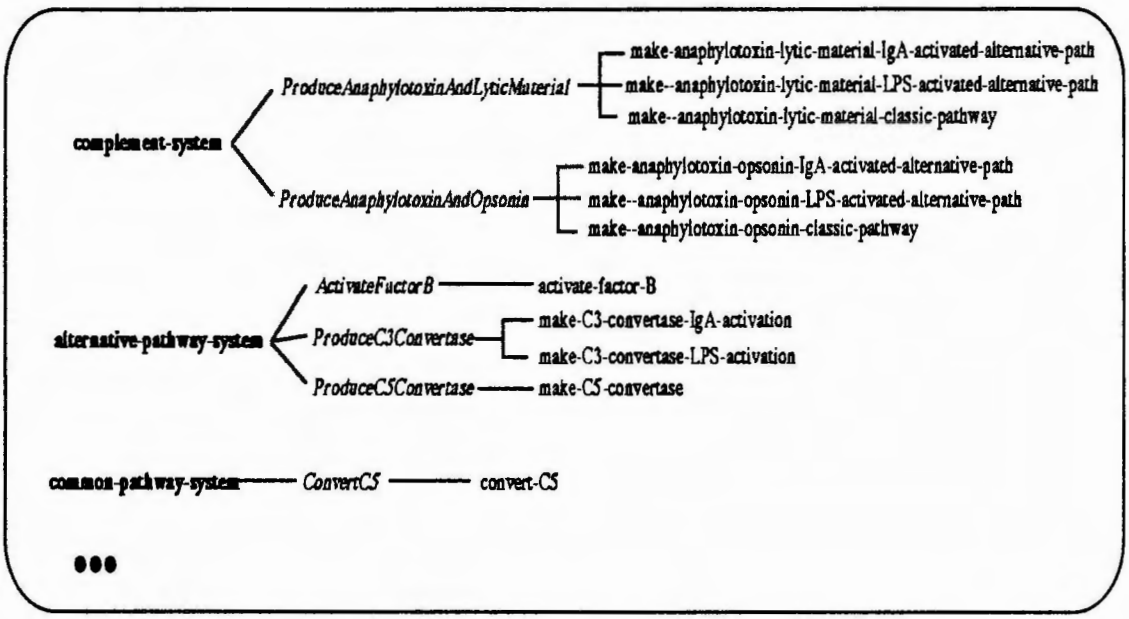


Figure 10: Component - Function - Implementing Behavior for a part of the Complement System

between anaphylotoxin and the Inflammatory Response will become important at a later point.

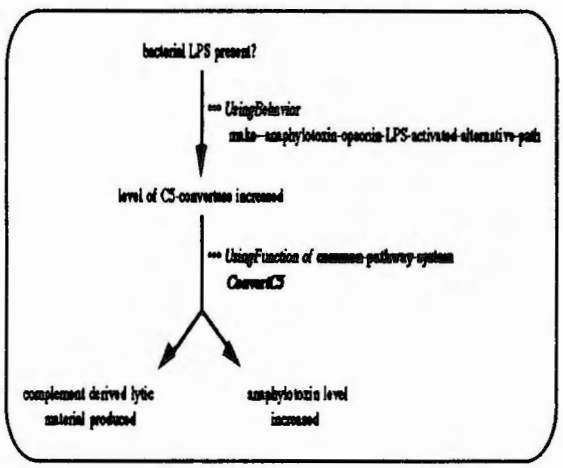


Figure 11: One Top Level Behavior of the Complement System

As before the link annotation indicates why, once C5-convertase is produced, it follows that anaphylotoxin and lytic material will be produced. In this case, the pointer is to the function ConvertC5 of the common pathway system.

Let us now go back to the link annotation leading to the presence of C5-convertase in Figure 11. This link annotation is one of the decomposable types, the UsingBehavior type. Hence, as we follow it, we come to an object having internal structure and, in this case, a behavior. The behavior pointed to, the "make-anaphylotoxin-opsonin-LPS-activated-alternative-path" behavior, is shown in Figure 12.

In Figure 12, we can see the fact that the presence of bacterial-LPS leads to the presence of C5-convertase because it produces C3-convertase as an intermediate step, and produces two materials that can be considered as side effects. These two side effects are the production of opsonin, and the production of anaphylotoxin. As mentioned earlier, anaphylotoxin is an instigator of the Inflammatory Response. Opsonin is a material which can coat bacteria, thereby marking it for identification by other parts of the Host Defense System. Neither material has direct use in the Complement System; hence the designation of functional side effects as concerns the

Complement System.

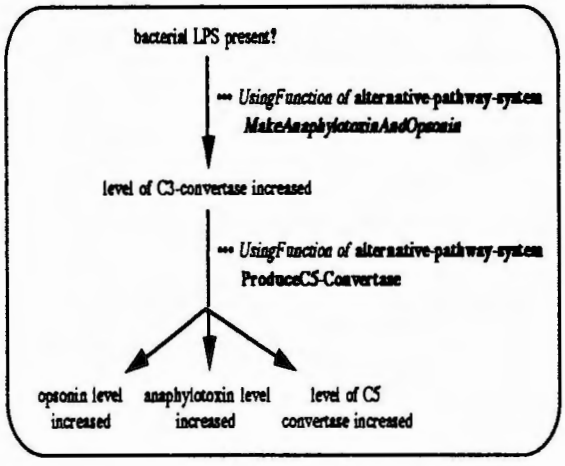


Figure 12: Second Level, Another Behavior

Now let us go on to more detailed levels of the representation. Suppose we follow the link annotation leading from the presence of C3-convertase in Figure 12. This is a pointer to the other type of decomposable knowledge source: a function. In Figure 13, we see the function ProduceC5Convertase function of the alternative-pathway-system.

The function shown in Figure 13 has the usual three part representation: the ToMake section tells us that this function's goal is to create c5-convertase, the By section points to the behavior which implements this function, and the Provided section tells us the predicates which must be true for this function to be applicable.

Note that in the less detailed behavior shown in Figure 12, there was one partial state which preceded the annotated link pointing to this function. This partial state basically was an assertion that C3-convertase was produced. The other starting conditions for the function of Figure 13; the presence of Complement Factor 3, the presence of ionized magnesium, and the presence of a blood factor called Properdin; are found only by following the knowledge pointer from the link annotation and examining the knowledge source it points to. These additional predicates should be thought of as the *assumptions* under which the function (in this case, and the behavior in other cases) operates.

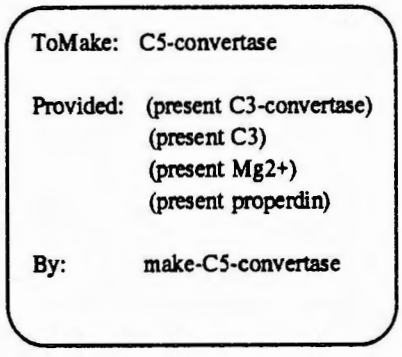


Figure 13: Third Level, A Function

Finally, let us follow the pointer to the implementing behavior pointed to in Figure 13 by the By clause. In Figure 14 we can see as though through a microscope many more intermediate states than were visible at the less detailed levels. Note as well that all of the Knowledge Pointers in the behavior of Figure 14 are of the non-decomposable type. This means that, in terms of our representation of the Complement System, we have reached the ground level.

We started in Figure 11 at the highest level of the complement system. At that level, it was easy to see what the overall functionality of the complement system was. But, given the very modular nature of the functional representation, it was also easy to follow one link annotation to another to deeper and deeper levels of detail in the system.

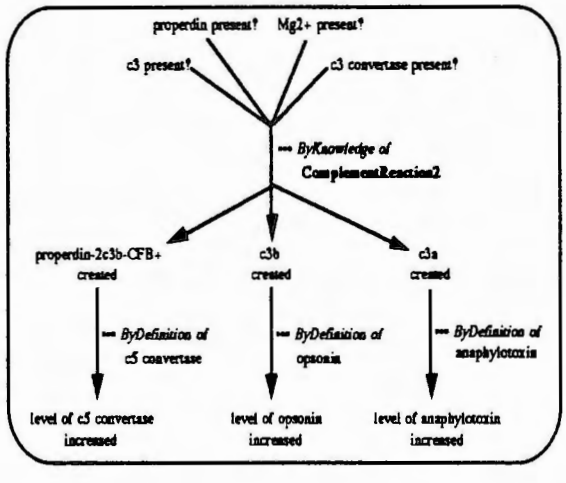


Figure 14: Most Detailed Level, Another Behavior

The exposition of the functional representation has followed to this point a "browsing" characteristic. In the next section, we will show how a functional

reasoner makes use of the representation we have shown above.

5.3. Reasoning about the Complement System

Suppose we consider a starting condition of (bacterial-LPS present). Let step through the method laid out in the Section 3.2.2 above describing the functional consequence finder to determine the effect on the complement system of the presence of bacterial-LPS.

5.3.1. Reasoning Step 1

The initial condition we set for this example is (bacterial-LPS present).

5.3.2. Reasoning Step 2

This step of the consequence finding strategy is aimed at determining which of the functions and/or behaviors of the Complement System are applicable given the initial conditions. The behavior shown in Figure 11 is one such behavior. However, indexing from (bacterial-LPS present) also would lead us to the behavior shown in Figure 12. Note that the Figure 11 behavior directly points to the Figure 12 behavior from its first level link annotation.

The first sub-step of Reasoning Step 2 is to index from the starting conditions to the behaviors and/or functions which are applicable. The second sub-step is to filter out those behaviors/functions which are either directly or indirectly pointed to by other members of the indexed set. Thus, because the Figure 11 behavior points to Figure 12 behavior, only the top level behavior of Figure 11 remains after the filtering sub-step. This illustrates the concept of "filtering" that is used in the functional reasoning algorithm.

5.3.3. Reasoning Step 3

Given the initial conditions, at this point in the overall reasoning process, we have produced the behaviors/functions which are the ones we should first consider. Reasoning Step 3 takes these initially indexed behaviors/functions and, by using the functional representation, produces a knowledge structure representing the state changes the device will go through *as a result of the stated initial conditions*. In the previous section, we called this knowledge structure the Particularized State Diagram (PSD).

Now following the rules of Step 3 of our algorithm, we would traverse the behavior of Figure 11. At each point in the traversal, we would expand any decomposable link annotations. The result of the complete traversal is shown in Figure 15. In the figure the PSD for this situation can be seen as a graph structure in which the nodes correspond to partial states in the behaviors from which they were derived.

The PSD as shown in Figure 15 contains all state transitions which are due to the initially indexed behaviors and *all* behaviors/functions those initial behaviors point to, either indirectly or directly. Once constructed, though, we may view the PSD at varying levels of detail. The lowest level of detail corresponds to the initial behaviors with all the knowledge pointers masked out. For the case we are examining here, the lowest detail level which we call Level 0 is shown in Figure 16. Note the direct correspondence with the behavior shown in Figure 11. The next higher level, Level 1, is shown in Figure 17; it corresponds to all partial states used in Level 0 and all those that are in behaviors/functions implicitly pointed to from the annotated links at Level 0.

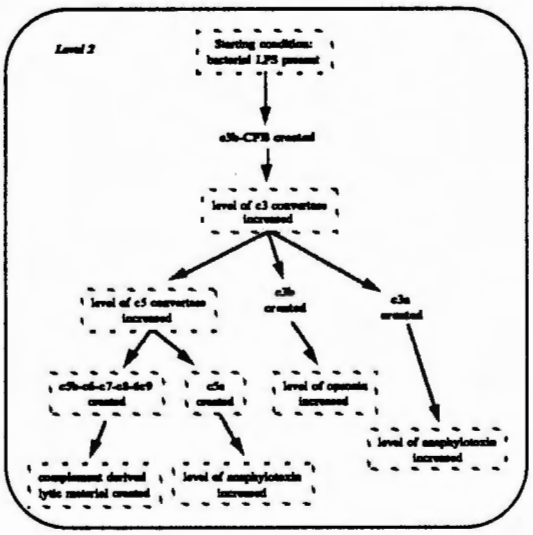


Figure 15: Consequence Finding - bacterial LPS - Level 2

Let us look more carefully at the PSD under the three "magnifications" shown in Figure 15, Figure 16, and Figure 17. At Level 0, in Figure 16, we see the initial condition, the presence of bacterial-LPS, followed by the presence of C5-convertase. Then, following the C5-convertase we see that lytic material will be produced and that anaphylotoxin will be increased.

Suppose we want to know why it is that C5-convertase will be followed by an increase in anaphylotoxin. There are two possible answers we can provide. We can answer based on the various assumptions for increased anaphylotoxin recorded internally in the PSD node. This way of answering involves laying out all the things that must be true in order for the increased anaphylotoxin to be true. It is important to note that the assumptions are *derived* by the process building the PSD.

We can also answer by showing the graph of Figure 17, at Level 1. In doing so, we answer by showing more detail, in that we are supplying intermediate partial states.

5.3.4. Reasoning Step 4

Once the PSD is constructed, it becomes relatively straightforward to determine what the consequences on the Complement System will be due to the presence of bacterial-LPS. We will require one auxiliary "scratch pad" - a simple database of values for the variables of state of the device. After initializing the database of state variables, we traverse the PSD making appropriate updates to the database at each node. Once completed, the consequences to the Complement System (actually, to the body) can be read off as the changes in the state variable database.

In addition to being able to determine these changes via the starting conditions, in this step our functional reasoner also derives the assumptions under which the state changes take place. This is because assumptions to get to each node are recorded in the node as the PSD is built. Thus it becomes easy to determine the cumulative assumptions from the completed PSD.

Figure 18 shows the output from our functional reasoner for the starting condition of (bacterial-LPS present).

The "Rule Compiler Invocation" number is the number of iterations through the steps of the functional reasoner. This is the first iteration through the steps (and the only one for this example), hence the number is 0. The "assuming" forms are the cumulative default assumptions. The "startingConditions" are obvious.

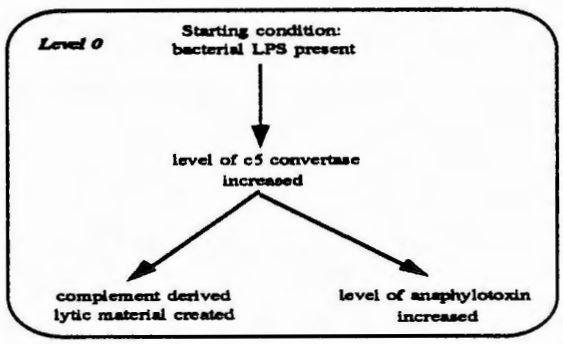


Figure 16: Consequence Finding - bacterial LPS - Level 0

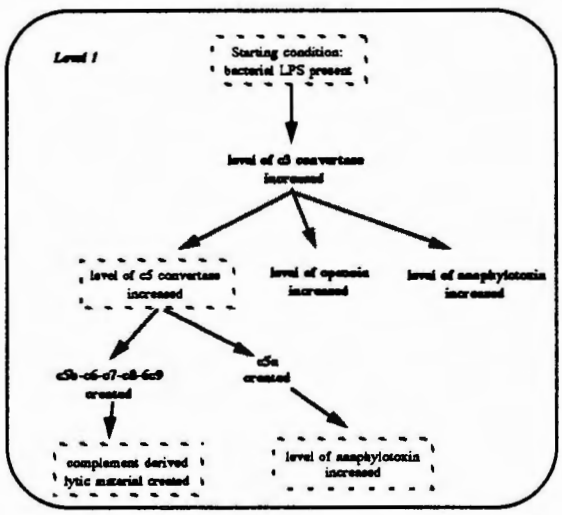


Figure 17: Consequence Finding - bacterial LPS - Level 1

The "Rule Compiler Invocation" number is the number of iterations through the steps of the functional reasoner. This is the first iteration through the steps (and the only one for this example), hence the number is 0. The "assuming" forms are the cumulative default assumptions. The "startingConditions" are obvious.

The "if" clause is the set of cumulative predicate assumptions which accumulate from each

behavior and function. The "then" clause is the set of cumulative changes for each class of state variable indicated by this PSD.

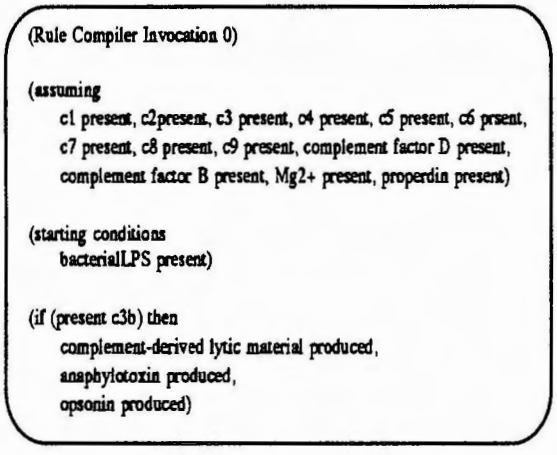


Figure 18: Output From Functional Reasoner given Bacterial-LPS

The effect of the presence of bacterial-LPS will result in the Complement System producing complement derived lytic material. But, there are a number of assumptions that must hold for this to be true, e.g.,

- assumptions about normal body components, such as C9, C8, ..., being available, and
- assumptions about material that must be made by a different system. In this case, the Complement System will not function properly unless some other component of the Host Defense System has broken down Complement Factor 3 to its constituent Complement Factor 3b.

Side effects of the production of lytic material include the production of anaphylotoxin, and the production of opsonin. The most important point about the result of this step is that assumptions under which associations between the starting condition and the resulting consequence are explicitly derived.

5.3.5. Reasoning Step 5

We now take the end result from Step 4; i.e., the cumulative changes to the device; and use them as starting conditions for a reinvokation of these Steps starting from Step 2. In this case, there will be no new behaviors/functions invoked; hence, the process will halt here with the final output being as shown in Figure 18.

In Section 4, we discussed the interaction between classification problem solving and function level consequence finding. In the case of bacterialLPS being present, anaphylotoxin is produced. Anaphylotoxin triggers the Inflammatory Response, which entails a number of observable signs and symptoms. Thus given an additional round of reasoning from the presence of anaphylotoxin to the manifestations of the Inflammatory Response, what we have accomplished is the derivation of a pattern which the classification hypothesis BacterialDisease can use to help determine its plausibility.

6. EASE OF DEVICE KNOWLEDGE ACQUISITION FOLLOWING THE FUNCTIONAL APPROACH

The functional representation of the human body complement system was implemented in a shell language tailored specifically for functional representation and reasoning. Once the shell language was completed, the actual implementation of the functional representation for the complement system required about one week. It was partially constructed by a computer non-professional - a medical faculty person.

The ease and speed with which the final implementation was completed support the value of the functional approach as a knowledge acquisition tool for device understanding. The approach is of course limited to those situations when one in fact knows the functionality of some object. If functionality is not known, and must be derived from scratch, then clearly the functional approach is not applicable.

Although the observation reported above concerning utility of the functional approach for manual knowledge acquisition is based only on informal experience, we can point to characteristics of the functional approach which are consistent with these observations. First, the functional approach is highly modular. The elucidation of causal knowledge about a device is not undertaken as one monolithic task following

the functional approach, but rather, causality is expressed in "bite sized" chunks.

Second, fragmentary causal knowledge indexed by abstract function should be easy to elucidate because it *is* indexed. During the process of manual knowledge acquisition, the basic process being undertaken is one of memory indexing. If the target structure for the acquisition has at its core the means by which indexing may be expressed, then the entire process should be eased.

Despite these and other arguments that seem to lend support to the idea that the functional approach would ease the burden of device knowledge acquisition, it is important that psychological investigations probing these claims be performed to rigorously test them. Until such studies are carried out, suggestive arguments such as in this section will be the best that can be done on supporting any claims for manual KA support.

7. DISCUSSION

We have described a functional approach to representing and reasoning with deep knowledge of devices, and shown how a functional reasoner can act to acquire new knowledge for a diagnostic problem solver. Although different in details and methodological basis, our goal to understand cooperative problem solving between compiled level diagnostic systems and deeper level problem solvers is somewhat similar to the recent work of Fink [Fink and Lusth 1987].

Several other lines of research have been reported which attempt diagnostic problem solving "from first principles", for example, [Reiter 1987]. This recent body of work is largely an extension of the diagnostic outlook taken by Davis [Davis 1984] and others that set out to perform diagnostic problem solving directly from a deep level. On the other hand, our approach integrates both a compiled level and a deep level problem solver into a *composite* unit in which the compiled level problem solver can play the important role of *focusing* the deep level problem solver.

The issue of problem solving focus also distinguishes our approach from the causal net approach. Once the "correct" part of a causal net is brought up for examination, then it is not too difficult to do very interesting problem solving based on it, as Patil for example does [Patil 1981]. But as was the case in the CADUCEUS system [Pople 1979], typically a causal net must be strongly augmented to be an efficient problem solver.[5]

The intuition of Patil that causal knowledge should be "layered" is however very compelling. Going back to the discussion of Section 5.3.3, and in particular to Figure 15, Figure 16, and Figure 17, it can be seen that a layered sense of causality is also part of the functional approach, but instead of being compiled in, it emerges from the natural combining of casual fragments during problem solving.

Finally, in relation to other lines of research, we point out that the naive physics research is simply aimed at solving a different problem than the one we attack. Research in that area attempts to derive large scale behavior, while we attempt to use it.

Beyond its uses in diagnostic problem solving, we are currently pursuing other independent uses for the functional modeling and reasoning methods we have discussed. In one project, we are applying functional techniques to modeling a complex agricultural ecosystem, with the ultimate goal being to better manage the system. In another project, we are applying functional modeling techniques to the fuel system of a modern high performance aircraft, with the goal being to manage the process of engineering the device, and in particular to utilize design knowledge in a trouble shooting context. Finally, we also hope to apply functional modeling techniques as a core part of an ICAI system to teach medical students human body physiology. In each of the domains we are exploring, the issue of knowledge acquisition is critical. Especially in the engineering domains, such as our aircraft fuel system, techniques must be developed which will support efficient KA. We are encouraged by our recent studies to believe that the functional approach will prove to be one such methodology.

5. In CADUCEUS the augmentation is largely "spanning links" which are compiled links in the causal net that helped to focus attention.

8. References

Abrett, G., & Burstein, M. H. (1986). The KREME Knowledge Editing Environment. Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop (AAAI). (pp. 1-1 to 1-19).

Bylander, T. (1986). Consolidation: A Method for Reasoning about the Behavior of Devices. Ph.D. Columbus, Ohio: The Ohio State University.

Bylander, T., & Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: The "Right" Level of Abstraction for Knowledge Acquisition. Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop (AAAI). (pp. 7-0 to 7-13).

Bylander, T., & Mittal, S. (1985). CSRL: A Language for Classificatory Problem Solving and Uncertainty Handling. AI Magazine. 7 (pp. 66-77).

Chandrasekaran, B. (1982). Decomposition of Domain Knowledge into Knowledge Sources: The MDX Approach. Proc. 4th Nat. Conf. Canadian Society for Computational Studies of Intelligence. Saskatchewan.

Chandrasekaran, B. (1983). Towards a Taxonomy of Problem-Solving Types. AI Magazine. 4 (pp. 9-17).

Chandrasekaran, B. (1985). Generic Tasks in Knowledge-Based Reasoning: Characterizing and Designing Expert Systems at the "Right" Level of Abstraction. Proceedings of The IEEE Second Annual Conference on Artificial Intelligence Applications. : IEEE.

Chandrasekaran, B. (1986). Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design. IEEE Expert. (pp. 23-30).

Chandrasekaran, B., & Mittal, S. (1983). Conceptual Representation of Medical Knowledge for Diagnosis by Computer: MDX and Related Systems. Advances in Computers(Yovits, Marshall). (22). (pp. 217-293). : Academic Press.

Davis, R. (1984). Diagnostic Reasoning Based on Structure and Behavior. Artificial Intelligence. 24 (pp. 347-410).

de Kleer, J., & Brown, J. S. (1984). A Qualitative Physics Based on Confluences. Artificial Intelligence. 24 (pp. 7-83).

Eshelman, L., Ehret, D., McDermott, J., & Tan, M. (1986). MOLE: A Tenacious Knowledge Acquisition Tool. Proceedings of the Knowledge Acquisition for Knowledge-Based Systems Workshop (AAAI). (pp. 13-0 to 13-12).

Fink, P. K., & Lusth, J. C. (1987). Expert Systems and Diagnostic Expertise in the Mechanical and Electrical Domain. IEEE Transactions on Systems, Man, and Cybernetics. SMC-17 (pp. 205-214).

Forbus, K. D. (1984). Qualitative Process Theory. Artificial Intelligence. 24 (pp. 85-168).

Hart, P. (1982). Directions for AI in the 80s. SIGART News Letter.

Kuipers, B. (1984). Commonsense Reasoning about Causality: Deriving Behavior from Structure. Artificial Intelligence. 24 (pp. 169-203).

Kuipers, B., & Kassirer, J. P. (1984). Causal Reasoning in Medicine: Analysis of a Protocol. Cognitive Science. 8 (pp. 363-385).

Marr, D. (1982). Vision. : W.H. Freeman.

Michie, D. (1982). High-road and Low-road Programs. AI Magazine. 3 (pp. 21-22).

Patil, R. S. (1981). Causal Representation of Patient Illness For Electrolyte And Acid-Base Diagnosis. Ph.D. Cambridge, MA: M.I.T.

Pople, H. (1977). The Formation of Composite Hypotheses in Diagnostic Problem Solving: An Exercise in Synthetic Reasoning. Proceedings of IJCAI 5. (pp. 1030-1037).

Pople, H. (1979). Heuristic Methods for Imposing Structure On Ill Structured Problems: The Structuring of Medical Diagnosis. P. Szolovits, Artificial Intelligence In Medicine. (pp. 119-189). : Westview Press.

Reiter, R. (1987). A Theory of Diagnosis from First Principles. Artificial Intelligence. 32 (pp. 57-95).

Sembugamoorthy, V., & Chandrasekaran, B. (1986). Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems. J. Kolodner, & C. Reisbeck, Experience, Memory, and Learning. : Lawrence Erlbaum Associates.

Sticklen, Jon (1987). MDX2: An Integrated Medical Diagnostic System. Ph.D. Columbus, OH: Ohio State University.

Sticklen, Jon, & Chandrasekaran, B. (1985). Use Of Deep Level Reasoning in Medical Diagnosis. Proc. of The Expert Systems in Government Symposium. McLean, Viginia.

Sticklen, Jon, Smith, J. W., Chandrasekaran, B., & Josephson, J. R. (1987). Modularity of Domain Knowledge. International Journal of Expert Systems: Research and Applications. 1 (pp. 1-15).

Weiss, S., Kulikowski, C., & Safir, A. (1977). A Model-Based Consultation System for the Long-Term Management of Glaucoma. Proceedings of IJCAI 5. (pp. 826-832).

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Semi-Automatic Acquisition of Conceptual Structure from Technical Texts

Stan Szpakowicz
Department of Computer Science
University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
szpak@csi2.uofo.edu

ABSTRACT

We present a system which semi-automatically processes technical text and incrementally builds the domain's conceptual model. Starting from an initial general model, knowledge-based text understanding turns into knowledge acquisition. Incompletely understood text fragments may contain new information to be integrated into the model under the operator's control. The text is assumed to fully describe the domain, and typical problems should be solvable by indicating activities which manipulate objects. Activities, objects and their properties enter relationships that form a conceptual network. To test our representation, we have created a large hierarchy of concepts for PowerHouse Quiz. The system, organized around the text and the growing network, includes a parser with broad syntactic coverage, and a matcher retrieving subnetworks relevant to the current text fragment. The frequency of the operator's necessary interventions depends on the initial network's size, to be determined experimentally. We discuss the status of the system and outline further work.

1. INTRODUCTION

This is a progress report on the development of an experimental system to process technical texts and incrementally build a knowledge base (KB) containing a conceptual model of the domain (Szpakowicz to appear). This project has evolved from two subsequent earlier projects related to knowledge acquisition from natural language texts (Skuce *et al.* 1985) and to knowledge representation (Szpakowicz *et al.* 1986). The system is based on the idea of turning the process of knowledge-based text understanding into knowledge acquisition. In the presence of a KB which contains a complete semantic representation of the domain, understanding a descriptive fragment of an expository text means relating it to this representation. If the representation is incomplete, full understanding may be impossible because of a gap in the KB. The missing information can be identified in the fragment and integrated into the KB.

The present state of the art in natural language processing does not yet allow fully automatic discovery and repair of gaps in KBs. The system we describe is intended to implement a *semi-automatic* process. This process involves an operator who steps in when the system cannot proceed on its own, and who monitors the system's decisions, overriding them if necessary. The system is only an intelligent assistant of a person who builds a KB from a text. In a sense, it learns about the domain, as knowledge accumulates during the construction.

Several basic assumptions have been made about the domain and the technical text. First, the domain does not require profound expertise to be mastered. Reading a technical text is sufficient to understand the domain and create its semantic representation. The participation of domain experts is not essential, although they still are needed in the preliminary and final phase of the KB construction. Examples of such relatively uncomplicated domains are software systems and application packages seen from the user's perspective. Their manuals usually contain an almost complete account of the domain. It is, however, a challenging task to recover the meaning from the linguistic form, typically rendered in a didactically oriented style; also, all texts about software presuppose certain minimal general knowledge of computing that must be represented within the same KB.

The KB that our system will build is a conceptual network, a variation of Sowa's conceptual graphs (Sowa 1984), with some simplifications allowed by the properties of the class of domains we consider. The network must be initialized as the domain's skeletal representation, built or at least validated by a domain expert. It is a matter of further research to determine what minimal knowledge is necessary to initiate the system without turning the process into an operation that would be entirely controlled manually.

The system accumulates knowledge in small steps, building on the fragments of the network accumulated so far. The source text is processed, in principle, sequentially. A crucial assumption about the composition of the source text is that the construction of a KB *can* proceed incrementally, by stepping through the text basically in the order it is written. This assumption seems justified in the case of technical texts with an overt didactic purpose, where the amount of backward references should be negligible and where forward references should concern details rather than new fundamental concepts. The system will access the text via a text base manager that allows the operator to follow references, to browse and to reach fragments related by topic or concepts to the currently processed fragment.

The next assumption: the source text is supposed to be linguistically correct, so that no special methods of parsing ill-formed structures need be applied. This seems to be the case for a majority of textbooks, user's guides and manuals, which are rather carefully edited. In those infrequent situations when the input is not entirely correct, partial analyses will be returned for the operator's consideration.

A skeletal KB for a typical domain has been developed as a test case. This domain is Quiz, part of PowerHouse, a fourth generation software package from Cognos Inc. The system is expected to work with a broad class of domains, but the prototype solutions are being tested only on a limited class of texts dealing with software systems of a specific variety. Another major assumption behind this research is that the findings from such prototyping will scale up to other domains.

The first version of the system will not handle examples, although they play an important role in expository texts, particularly in tutorials and manuals which frequently use examples to introduce new information. The treatment of examples in technical texts is a separate research problem, closely related to learning. Here we are primarily concerned with descriptive text.

The rest of the paper is organized as follows. We first discuss the method of knowledge representation we have chosen for our system. Next, the overall design of the system is presented, followed by a more detailed description of the proposed acquisition procedure. In the next two sections we summarize the present status of the system, and we review some related work. A paragraph on further research plans concludes the paper.

2. KNOWLEDGE REPRESENTATION FOR SOFTWARE DOMAINS

The system we describe will process texts from domains typified by fourth generation software systems and packages. An expert system or question-answering system built around a KB constructed by our procedure will be expected to solve problems characteristic of the domain, usually by indicating one or more highly specialized activities operating on domain objects. Domain concepts, classified into activities, objects and properties of objects, are involved in binary relationships, both general (for example, roles of objects in activities; spatial relations between objects; temporal relations between activities) and those specific to the domain. Concepts are also organized into the usual hierarchy of specialization (*is_a* links) and generalization (*kind* links).

Concepts handled by the system will represent only generic knowledge about typical activities and objects. Facts about specific situations, necessary to answer specific questions (for example, how to diagnose an error signalled in a command or how to explain an unexpected result of a query) can be derived from generic facts by instantiation.

It should be pointed out that we have attempted to make our representation as simple as possible, without compromising its expressive power. It has been intentionally restricted in several ways. Each concept must be unequivocally classified as an activity, an object or an abstraction, and events other than activities are not representable;

relationships between more than two concepts are expressed in an indirect manner, as described below; specific knowledge cannot be represented. A proper evaluation of these design decisions should involve a thorough theoretical analysis that would take into account inheritance, quantification, causality and so forth. Meanwhile, experience shows that all these subtle phenomena are readily expressed in our notation, restrictions notwithstanding.

A collection of mutually related concepts is represented as a conceptual network, a variant of Sowa's conceptual graphs, with the emphasis on activities and objects. We have developed a notation in which every concept is associated with a cluster of concepts directly linked to it by binary relationships. We call such a cluster a *frame*.

A concept is fully identified by its name and structure: *individual*, *sequence*, *set* or *tuple*. The frame notation allows us to group everything that is related to the given concept: an activity is described together with a sequence of identical activities, an object together with a set of objects, and so on. For example, the same frame describes a *file* which is the result of some activity and a *set of files* which are the data for another activity. This organization helps increase the correlation between concepts and the text being processed: the conceptual network is accessed from the lexicon which associates concepts with words rather than phrases, and further processing takes advantage of the knowledge already stored in the network.

Relationships between three or more concepts are represented as groups of binary relationships, mutually linked by the referential identity of concepts. For example, if an object B is related to two activities, A1 which produces B and A2 which uses it, then A1 and A2 are indirectly related through B. More subtle links are described by conditions attached to descriptions of relationships. Conditions place various contextual restrictions on the concepts involved in a relationship. A precondition must hold for the relationship to be meaningful. A postcondition, which may only be attached to an activity, specifies its effect on the existence and properties of objects it affects. For example, an object may be suitable as data of an activity only if it has some special property (the object might be, say, an input device, and the property might be *open for reading*). Another example is a restriction on the cardinality of a set of objects collectively involved in an activity, such as the fact that, in a relational database, it takes at least two relations to produce a join.

Concepts involved in conditions can be denoted indirectly by paths. A *path* describes a concept as related to another concept which, in turn, may be denoted by a path. For example, the condition

source of selecting_with_conditions == object of accessing

states that certain generic concept (in this case, a sequence of files) is related both to

selecting_with_conditions as its *source* and to *accessing* as its *object.* This should be understood as a restriction on the eventual instantiation of the generic concept.

Another form of a path is an application of a *selector* to a concept which must be a sequence, a set or a tuple. For example,

first of object of accessing

means the first of a sequence of files which are the object of the activity *accessing.*

A word on the notation: a frame for concept C_0 describes a cluster of concepts C_1, ..., C_k directly related to C_0. Each C_i is described in a slot which has the form

[precondition]
(structure_of_C_0) relation_name : (structure_of_C_i) C_i
[postcondition]

Both structures are optional - they default to *individual.* Conditions are optional - they default to true. The symbol *this_frame* refers to C_0, the symbol *this_slot* to C_i described in the current slot. Thus, the slot

[not one of this_slot == result_file] object: (sequence) file.

in the frame for *accessing* means that the object of accessing is a sequence of files, provided that it is not true that any of these files is a *result_file.*

Here is a slightly larger example - the description of accessing and its two major subclasses.

```
ACTIVITY accessing.
is_a: single_act.
kind: single_file_accessing.
kind: linking.
result: (sequence)record.
[not  one of this_slot == result_file]
   object: (sequence) file.
end accessing.

ACTIVITY single_file_accessing.
is_a: accessing.
[length of object of this_frame = 1]
   result: (sequence) file_record
      [status of first of object of this_frame == primary_file].
end single_file_accessing.
```

```
ACTIVITY linking.
is_a: accessing.
[length of object of this_frame >= 2 ]
   result: (sequence) record_complex
      [status of first of object of this_frame == primary_file and
       status of each of tail of object of this_frame == subordinate_file].
end linking.
```

We have implemented a translator of frames into a Prolog data base. The translator validates each relationship and all the paths mentioned in conditions. A simple inheritance mechanism is used; for example, the *object* of *linking* in the frames above is inherited from *accessing*. The translator discover pairs of relationships which may be mutually inverse (*is_a* and *kind* are an example). Pairs accepted by the operator are subsequently recognized: when one of the pair is defined, its inverse will be automatically added. Incomplete paths are recorded and may be stored later, once their missing fragments have been defined; this recording mechanism makes it possible to process the input in one pass.

The approach and notation discussed above have been tested on one domain, Power-House Quiz. A hierarchy of about 300 concepts has been created manually to allow experiments with the incremental construction of the KB; in Figure 1 we show two fragments of this hierarchy, displayed by a graphics module of the preliminary user interface.

The hierarchy does not yet include many necessary general computing concepts, but places have been designated to connect sub-hierarchies of such concepts to the existing hierarchy. Nirenburg and Raskin (1987) suggest one such an *is_a* hierarchy for "the computer science world".

The findings and experience accumulated during work on Quiz are immediately applicable to similar domains, namely to command-oriented fourth generation languages. Other software domains, such as operating systems (seen through their command language rather than their internal structure), should be relatively easy to describe in the same framework.

3. ORGANIZATION OF THE SYSTEM

The system has been designed with six major modules organized around the text base, which contains the source text, and the incrementally growing conceptual network. The overall organization of the system is shown in Figure 2.

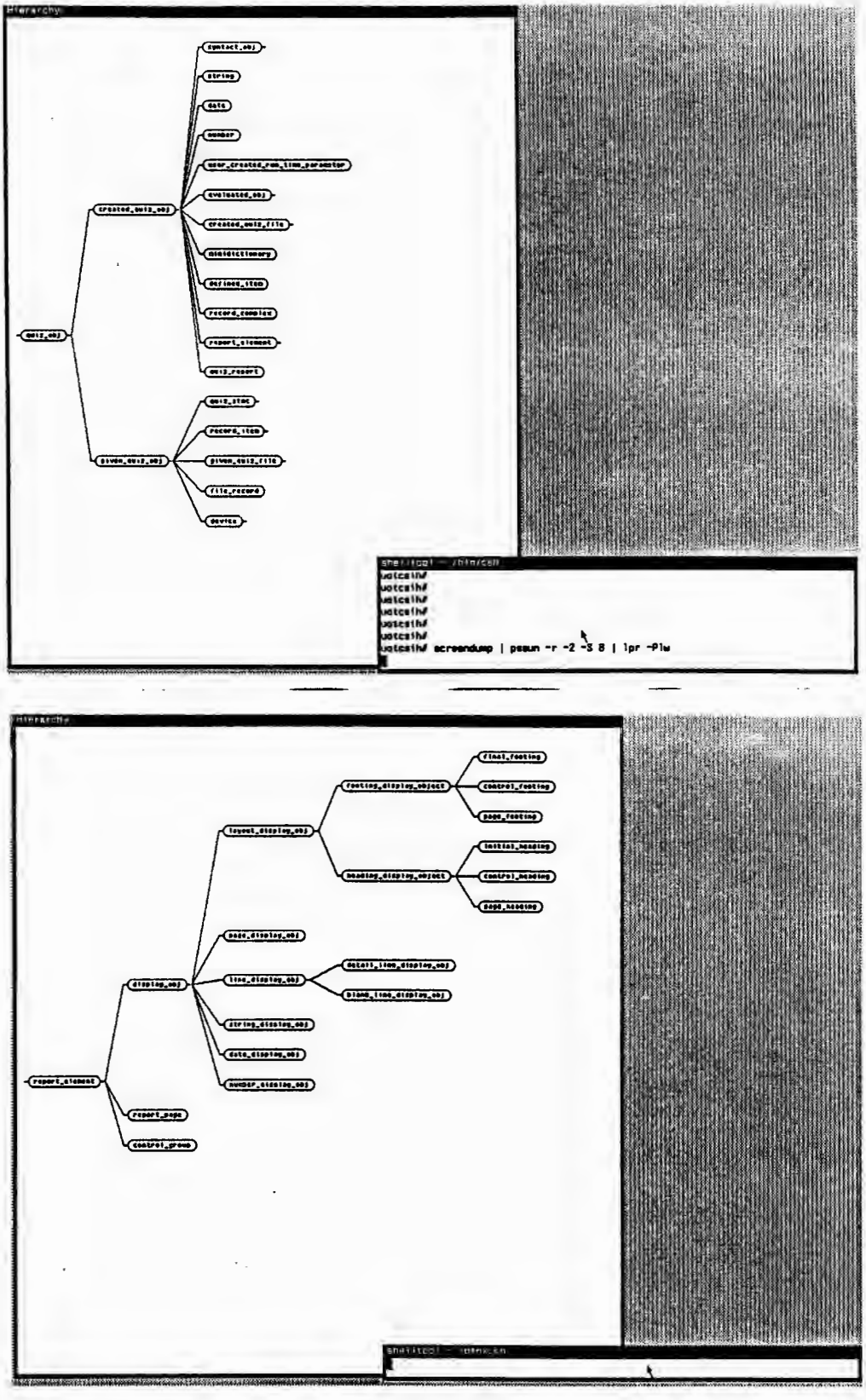


Fig. 1. Fragments of the hierarchy of Quiz concepts

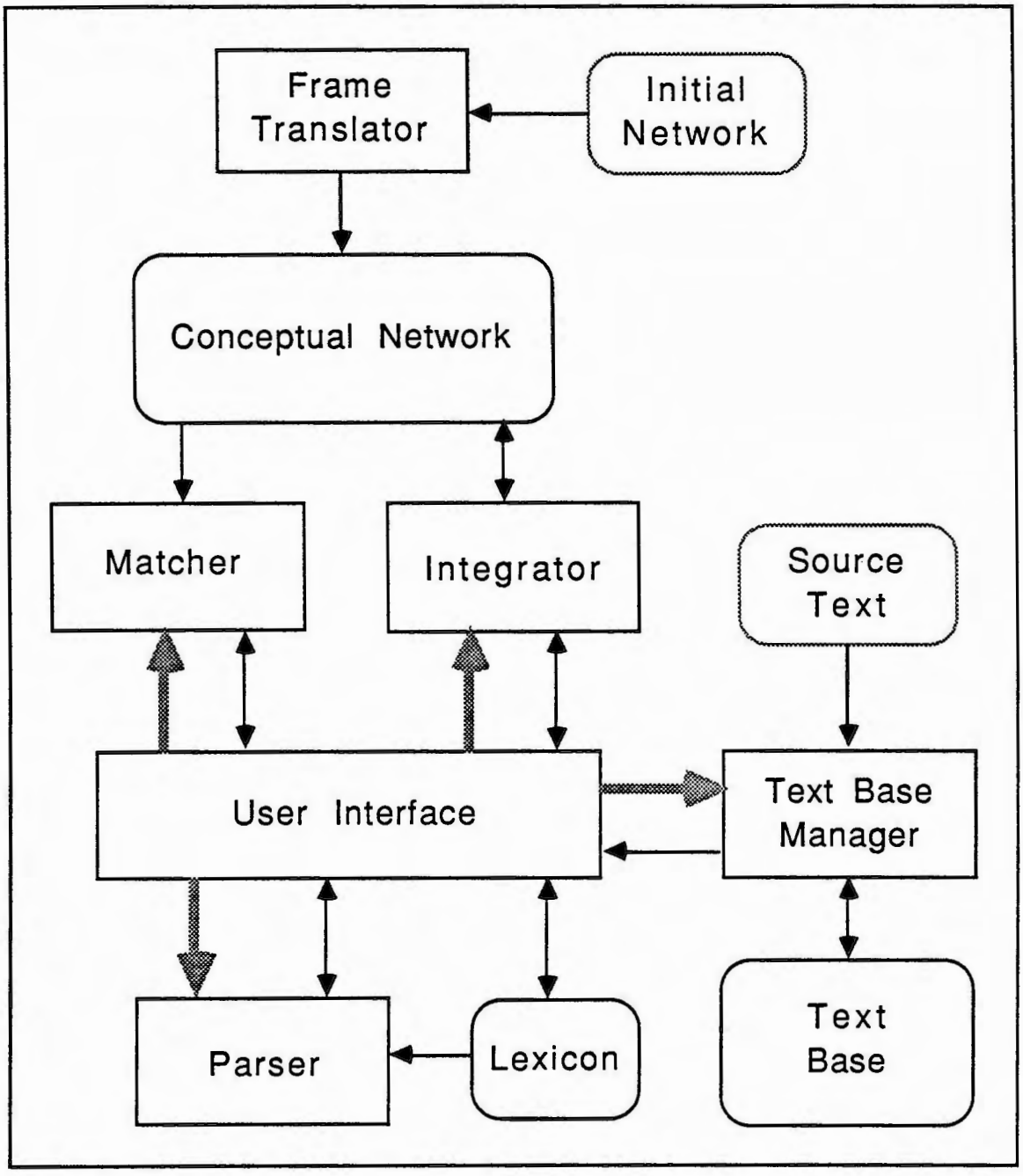


Fig. 2. Organization of the system

The user interface will communicate with three sources of information: the user, the network kept in the knowledge base and the text base (via the text base manager). It will activate the remaining modules to analyze each incoming sentence of the source text.

The heart of the system should be a parser capable of processing most of the source sentences. The parser currently analyzes a sentence into an intermediate surface-syntactic structure such as that shown in Figure 3. Some sentences may not carry any meaningful information but serve a useful didactic purpose. In structures for meaningful sentences, verbs will be associated with activities and noun phrases with objects playing various roles in activities. The parser will be extended to maintain simple contextual information about the current topic in order to help resolve some anaphoric references. The lexicon used by the parser is also accessed by the user interface when a new word is added interactively during processing.

The semantic phase of the parser will transform the intermediate structure into a conceptual representation, a collection of references to the KB; see Figure 4 for an example. Depending on the completeness of the analysis, this collection may be as small as one concept or as large as a mini-network which can be identified as part of the KB. Fragments of the sentence unaccounted for by semantic mapping will be given to the operator for consideration.

If the results of semantic mapping are meaningful enough to attempt their integration with the KB, the matching module will be invoked. It tries to match the mini-network with the entire network accumulated so far, and to retrieve its relevant fragments; see Figure 5 for an example. If this fails because of the insufficiency of the mini-network (which in an extreme case may be reduced just to a few concepts), associations are sought between concepts recognized in the sentence. Paths in the network as well as indirect connections through conditions are identified. These paths are then combined into a tentative mini-network which is once more matched against the entire network. As a result of this, information will be isolated that has not yet been recorded in the KB.

A separate module (not yet implemented) will be responsible for the actual integration of the network and new information found in the current text fragment. Normally such integration should consist in adding new nodes to the network. Problems that may arise in the case of conflicts between old and new knowledge will be a matter of further research.

The last module of the system is the translator from the external notation for frames into the internal format of the KB. It is used to initialize the network.

```
sent(simple_decl(
stmt(
  agent(
    entity(
      n(dictionary, classf([])),
      sg, det_seq([the]), [], nil)),
  predicate(
    act(v(contain, [pres3s], neg(yes), adverb(nil))), active,
    cpl(entity(
      n(information, classf([])),
      sg, det_seq([the]), [],
      rel_cl(
        rel_m(nil),
        stmt(
          agent(
            entity(quiz, sg, def, [], nil)),
          predicate(
            act(v(need, [pres3s], neg(yes), adverb(nil))), active,
            cpl(
              entity(
                n(information, classf([])),
                sg, det_seq([the]), [])),
            modifiers([
              inf_clause(
                predicate(
                  act(v(provide, [inf], neg(yes), adverb(nil))), active,
                  cpl(ref(you, sg/pl)),
                  modifiers([
                    pp(with,
                      conj(and, 2, [
                        entity(
                          n(heading, classf([page])),
                          pl, det_seq([]),
                          [attr(automatic, pos, advp([]))]),
                        entity(
                          n(feature, classf([])),
                          pl, det_seq([many, other]), [])], nil))]))])))),
          sent_mods([])))))),
    modifiers([])),
  sent_mods([adv(also)])))
```

Fig. 3. The intermediate structure for the sentence

"The dictionary also contains the information QUIZ needs
to provide you with automatic page headings
and many other features."

```
[ edge(     concept( sorting, individual, 'ACTIVITY' ),
            Role1,
            concept( item, set, 'OBJECT' ) ),
  edge(     concept( selecting, individual, 'ACTIVITY' ),
            Role2,
            concept( condition, set, 'OBJECT' ) ),
  edge(     concept( selecting, individual, 'ACTIVITY' ),
            instrument,
            concept( 'AND', Struct1, 'OBJECT' ),
  edge(     concept( 'AND', Struct1, 'OBJECT' ) ),
            is_a,
            Parent1 ),
  edge(     concept( selecting, individual, 'ACTIVITY' ),
            instrument,
            concept( 'OR', Struct2, 'OBJECT' ),
  edge(     concept( 'OR', Struct2, 'OBJECT' ) ),
            is_a,
            Parent2 ) ]
```

Fig. 4. A possible conceptual representation, in Prolog, of the sentence

"Just as you can SORT on several items, so you can SELECT on several conditions, using AND or OR."

```
concept( sorting, individual, 'ACTIVITY' ) --->
consists_of --->
concept( sequencing, individual, 'ACTIVITY' ) --->
inv( instrument ) --->
concept( item, sequence, 'OBJECT' ) --->
struct_similarity --->
concept( item, set, 'OBJECT' )

concept( selecting, individual, 'ACTIVITY' ) --->
kind --->
concept( selecting_with_conditions, individual, 'ACTIVITY' ) --->
instrument --->
concept( condition, individual, 'OBJECT' ) --->
struct_similarity --->
concept( condition, set, 'OBJECT' )
```

Fig. 5. Examples of paths found by the matcher for the conceptual representation of Fig. 4

4. DETAILS OF THE TEXT PROCESSING PROCEDURE

The system will be initialized with a skeletal KB containing the top level of the conceptual network of the domain. The initial lexicon contains the basic vocabulary: a few hundred closed-category words (articles, prepositions, pronouns, conjunctions, numerals) and open-category words - nouns, verbs, adjectives, adverbs - related to the concepts already present in the skeletal KB. Open-category words point at the nodes of the network. The parser receives a sequence of sentences, some of them incomplete because of the omission of examples. Examples will be suppressed by the text base manager but displayed along with the sentence so that the operator can see the complete original text.

The text can be processed in three modes. In the *sentence mode*, already partially implemented, sentences are given to the user interface one by one, although the operator will always be able to reach any close or remote fragment and inspect it, usually without analyzing it thoroughly or at least without changing the KB. In the *paragraph mode*, a larger fragment (often a paragraph) will be analyzed at once, and the resulting mini-networks accumulated. Only after the paragraph has been analyzed will the system try to integrate its semantic representation with the existing KB. Finally, the *free mode* will allow the operator to choose any fragment and try to add its contents to the knowledge base. The text base manager will mark such fragments as processed.

The lexical analyzer prompts the operator for descriptions of words not yet present in the lexicon. Parsing and semantic analysis will proceed as described in the previous section, with meaningless sentences displayed and then skipped. If a sentence cannot be parsed (usually because of an omission in the parser), partial analyzes will be shown to the operator who will be able to resubmit the sentence in a slightly altered form. For example, the parser should indicate a situation when a sentence contains an expected case marker (such as a preposition typically introducing an instrument of an activity) with an unexpected filler (such as a location).

The conceptual mini-network built for a fragment - most often, a sentence - will be matched against the KB. If the matching succeeds, the fragment will not contribute new information. An expository text may contain the same information in several places, for example, in a tutorial and later in a reference section.

Mismatches of two kinds are possible. First, the fragment may not be completely understood by the system, but the relevant concepts and relationships have been recorded in the KB. The operator should recognize this situation by examining the relationships retrieved during matching. Second, part of the mini-network cannot be found in the KB. This will mean that new information is to be added, and the integration module invoked. (This happens when the fragment refers to known objects with new properties, or to new objects somehow related to known activities. New concepts

will usually belong to the lower levels of the concept hierarchy, the skeletal KB being already in place.) Again, the operator will be responsible for activating this integration.

The rate at which this process will augment the KB without too numerous interventions of the operator will depend on the size of the initial network. It must not be too detailed, but it should be large enough to allow smooth integration of new knowledge, acquired from successive fragments of the text. The network already created for Quiz seems large enough to allow experiments with various cut-off points, once a complete prototype of the system has become operational.

5. CURRENT STATUS OF THE SYSTEM

We have carried out linguistic analysis of the tutorial part of the Quiz manual as a typical example of text to be processed by the system (Szpakowicz in preparation). The vocabulary turned out to be rather modest in size, and the complication of syntactic structures relatively low. We found that the parser becomes much smaller and easier to develop if certain difficult linguistic problems can be solved by the operator. Examples of such problems are: subtle anaphora resolution, more complete treatment of apposition, recognition of idiomatic expressions. The parser may propose tentative solutions but the operator will be expected to make decisions. The handling of ellipsis will be left entirely to the operator, because, as the analysis of the Quiz manual suggests, it may be infrequent in technical texts.

A surface-syntactic analyzer with broad coverage has been designed and implemented in Prolog. It processes a majority of sentences from the tutorial part of the manual. We expect that with a fully debugged parser only a small fraction of the analyzed sentences will require manual intervention. The analyzer is currently being extended into a case-based parser that will produce a crude conceptual representation. Refining this representation will be preceded by more work on the parser to separate conjoined clauses and phrases (already accepted by the parser) into fragments directly corresponding to domain objects and activities.

The complete text of the Quiz manual has been acquired and is available on-line. The text base manager is now being implemented. It will divide the text into elementary units: sentences or sentence fragments, Quiz code, tables, placeholders for graphics. It will also perform most of the low-level lexical preprocessing.

A conceptual model of Quiz has been created. It is represented as a large network comprising about 300 concepts. The frame translator, used to initialize the network, is fully operational. It sets up a KB which contains the Prolog representation of the network and a number of auxiliary relations that facilitate access to its parts.

The matching module works for the time being only on sets of concepts, because the parser does not yet produce coherent mini-networks. The tested part of the matcher finds associations between concepts and identifies a sub-network spanning these concepts. The details of the matcher's algorithm will be described in a forthcoming report.

All the implemented software runs in Quintus Prolog 2.0 on a Sun-3 workstation.

6. RELATED RESEARCH

References to work on language analysis with the purpose of accumulating knowledge are scarce, and there are no mentions of semi-automatic approach similar to one described here. Frey *et al.* (1983) describe a system which translates sentences in restricted German into Prolog clauses. Complete texts are processed automatically, using a special intermediate discourse representation. Alshawi (1985) presents a system which incrementally builds a conventional database from rather simple English sentences, using a single knowledge base of linguistic and domain-specific facts.

Adam and Fargues (1984) present a design rationale of a system that will acquire knowledge from French texts. Interaction with the user is expected to supplement parsing and provide missing information necessary to understand arbitrary texts. Sowa's conceptual graphs were chosen as the final translations of individual sentences (Fargues *et al.* 1986).

Berry-Rogghe and Kaplan (1986) directly address the problem presented in this proposal. They consider, in general terms, the macrostructure of expository texts and outline a highly recursive text grammar and a text-to-knowledge transformation algorithm, both formulated at a very abstract level. The authors concede that their paper is just a preliminary proposal. They do not even consider how much manual intervention may be required to achieve any practical results.

Tucker *et al.* (1986) outline a basis for modelling discourse cohesion in expository texts. Their results are applied to machine translation. Nishida *et al.* (1986) concentrate on intersentential relations and various levels of headings which they propose to use to organize knowledge extraction from texts.

Diederich (1987) mentions text analysis as one of the methods of knowledge elicitation in a hybrid tool, but the emphasis is on the variety and integration much more that on linguistic issues.

Moreland and Selfridge (1986) present a theory and computer implementation of automatic processing of news stories, based on Schank's theories of Conceptual

Dependency and Dynamic Memory. While this proposal concentrates on human-assisted KB construction from a large text, the emphasis of their work is on in-depth understanding of stories by accumulating facts read or inferred from text and then answering questions. Nevertheless, the long-term research goals are very similar.

7. FUTURE WORK

The project has gained momentum only recently, so that it is early to fully evaluate its merits. It should take a year to finalize a pilot version of the system. Meanwhile, experiments with the already completed modules and with high-level prototypes of the others have been encouraging, although the main result of these experiments has been the realisation that much more work remains before any measure of success can be claimed. Future work will concentrate on improving the quality of linguistic analyses and the accuracy of the matching algorithm, and on finding the optimal size of the initial network. As another important experiment, the system will be applied to other domains. Finally, a knowledge base created by our procedure should be used in the construction of an actual expert system, so as to demonstrate that this is a viable method of overcoming the knowledge acquisition bottleneck.

ACKNOWLEDGEMENTS

Thanks to Lyne DaSylva, Zbig Koperczak, Bogdan Materna and Yannick Toussaint who helped implement various parts of the current prototype. This work has been supported by the NSERC grant A8193.

REFERENCES

Adam, J. P., Fargues, J. (1984), "Project KALIPSOS, General Overview". Internal research Report F074, IBM Paris Scientific Center.

Alshawi, H. (1985), "Creating Relational Databases from English Texts". *Procs 2nd IEEE Conference on Artificial Intelligence Applications*, pp. 449-454.

Berry-Rogghe, G., Kaplan, R. M. (1986), "On the Structure of Expository Texts; Preliminaries to Building an Expert System from Manuals". In: Boose, J. H., Gaines, B. R. (eds.), *Procs Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff. Also published in *Procs 6th International Workshop on Expert Systems and Their Applications*, Avignon, pp. 1125-1132.

Diederich, J. (1987), "Knowledge-based Knowledge Elicitation". *Procs 6th National Conference on Artificial Intelligence AAAI87*, pp. 201-204.

Fargues, J., Landau, M.-C., Dugourd., A., Catach, L. (1986), "Conceptual graphs for semantics and knowledge processing". *IBM J. Res. Develop.*, vol. 30, no.1.

Frey, W., Reyle, U., Rohrer, C. (1983), "Automatic Construction of a Knowledge Base by Analyzing Texts in Natural Language". *Procs 8th Int'l Joint Conf. on Artificial Intelligence IJCAI 83*, Karlsruhe, pp. 727-729.

Moreland, B. L., Selfridge, M. (1986), "Acquiring Business and Environmental Knowledge from Real-World Natural Language Text". Univ. of Connecticut, Dept. CS and Eng. TR 86-15.

Nirenburg, S., Raskin, V. (1987), "The Subworld Concept Lexicon and the Lexicon Management System". *Computational Linguistics*, vol. 13, no. 3-4, pp. 276-289.

Nishida, F., Takamatsu, S., Tani, T., Kusaka, H. (1986), "Text Analysis and Knowledge Extraction". *Procs 11th International Conf on Computational Linguistics COLING '86*, Bonn, pp. 241-243.

Skuce, D., Matwin, S., Tauzovich, B., Szpakowicz, S., Oppacher, F. (1985), "A Rule-Oriented Methodology for Constructing a Knowledge-Base from Natural Language Documents". In: K. Karna (ed.), *Expert Systems in Government Symposium*, IEEE, pp. 378-385.

Sowa, J. (1984), *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley.

Szpakowicz, S., Matwin, S., Skuce, D. (1986), "QUIZ Advisor: A Consultant for a Fourth Generation Software Package". *Procs 6th International Workshop on Expert Systems and Their Applications*, Avignon.

Szpakowicz, S. (to appear), "A Semi-automatic Procedure for Knowledge Acquisition from Technical Texts". Procs Symposium *Textes et Reprèsentations*, Quèbec, April 1987, Presse Universitaire de Laval.

Szpakowicz, S. (in preparation), "Linguistic Coverage of a User's Manual for a 4GL Language".

Tucker, A. B., Nirenburg, S., Raskin, V. (1986), "Discourse and Cohesion in Expository Text". *Procs 11th International Conf on Computational Linguistics COLING '86*, Bonn, pp. 181-183.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Performance-Based Knowledge Acquisition: A Structural Foundation for Traditional Introspective Knowledge Acquisition

DRAFT

Deborah Tranowski, Tod Levitt, Kaitilin Riley

We propose that two complementary methods of knowledge acquisition can provide more structure to standard techniques and provide statistically-viable evidence in support of acquired rules and other knowledge. Two knowledge acquisition (KA) approaches being used are:

- Introspective techniques: interviews with experts
- Performance-based techniques: having experts accomplish their tasks and analyze their performance to determine the key knowledge they use

The fundamental difference between introspective versus performance-based knowledge acquisition is whether the heuristics/rules that explain the expert's behavior at his domain task are elicited directly from the expert, by asking him to introspect on his task performance, or whether the rules are inferred by the knowledge engineer after observing the expert's performance at his task. The latter method is sometimes used by decision theorists, who often use statistical tools to infer relationships in the knowledge elicited from domain experts. (See [Von Winterfeldt and Edwards - 86], [Efron - 82]). Because decision theorists have not traditionally gone on to infer declarative rules explaining the expert's behavior, we are actually extending the traditional approach. The introspective knowledge acquisition approach is the one traditionally used by expert system builders, ([Hoffman - 87], [Hayes-Roth - 83]).

Reasoning primitives form the vocabulary in which domain experts discuss their domain. The observation of these primitives may involve significant cognitive processing on the part of the expert (refer to Figure 1). For example, reasoning primitives might include observation of symptoms for medical diagnosis, measurements in seismic data for oil exploration or as in our application image processing to segment an image into meaningful pieces. We see that reasoning primitives comprise the vocabulary in which experts will express the heuristics, rules and inference strategies they use to perform their tasks.

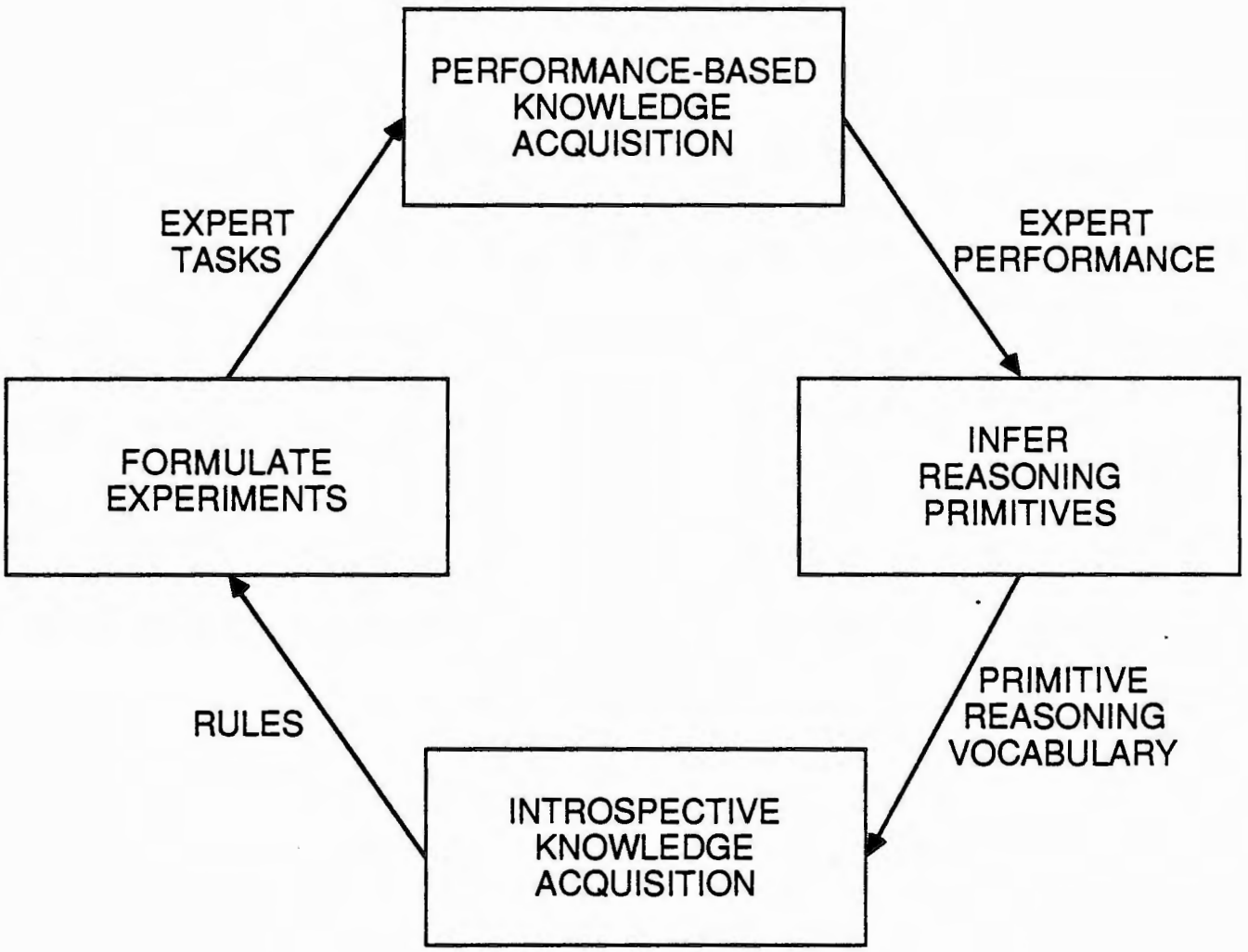
We applied the two methods of knowledge acquisition to infer the reasoning primitives used by photo-interpreters to analyze the content of imagery. The goal is to identify the contextual aids used by photo-interpreters in performing scene analysis. Contextual aids are any non-imagery derived information that can support the image exploitation process (e.g., terrain cues, environmental information, vehicle spatial layouts).

KA techniques are utilized in a complementary fashion for acquiring the contextual aids used by photo-interpreters. As shown in Figure 1, during the introspective KA sessions, the domain expert and knowledge engineer interactively discuss the interpretation strategies and contextual aids the domain expert uses. The responses elicited from the domain experts are documented and used to form hypotheses and experiments that set the stage for the performance-based KA sessions. During the performance-based KA, photo-interpreters are presented with a task to perform that closely corresponds to tasks a photo-interpreter normally performs. The performance-based KA experiments are designed to evaluate some

of the hypotheses (responses) derived from the introspective KA. A combination of heuristic-guided statistical analysis, inductive learning techniques, and methods from decision theory are used to analyze the performance data in light of the contextual aids. Any significant deviations between the introspective ("what the expert said he does") and performance-based ("what the expert actually does") are noted and discussed.

We are conducting one introspective KA session and a corresponding performance-based KA session once a month. The KA sessions started in February, 1988 and will be completed in August, 1988. Results will be reported at that time.

FIGURE 1: PERFORMANCE-BASED AND INTROSPECTION KNOWLEDGE ACQUISITION CYCLE



3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

Knowledge Acquisition at the Front End: Defining the Domain

Brian Woodward
Knowledge Sciences Laboratory
Department of Computer Science
The University of Calgary
Calgary, Alberta, Canada

Abstract

The Knowledge Engineering process may be partitioned into three general phases. The first phase is characterized by domain definition, application selection and resource commitment. The second reflects the numerous iterations of knowledge acquisition and knowledge-base development. The final phase is the installation, maintenance and use of the system. Unlike phase two, which is characterized by powerful methodologies, few structured techniques for domain definition exist for the first phase. Knowledge Acquisition principles, but not current knowledge acquisition tools, are applicable to phase one activity. Cognosys is a semi-automated system developed to build general content models of a domain before the use of specific knowledge acquisition tools. Cognosys is described, as are future developments and research using the tool.

Introduction

The rapid growth, development and use of knowledge-bases is largely due to powerful Artificial Intelligence methodologies for knowledge representation and knowledge processing. The "engineering" of knowledge into useful knowledge-based systems requires more than these methodologies: it requires procedures for acquiring information and knowledge in such a manner as to both accurately reflect the original human knowledge and to maintain its integrity when replicated in a machine environment. The activity of knowledge acquisition has grown rapidly to accommodate and facilitate the process of knowledge transfer from human knowledge sources, such as a domain expert, to a machine usable representation (Gaines, 1987).

The process of knowledge-base development (Figure 1) begins with the initial selection of the problem, application or domain definition; moves through iterations of knowledge acquisition activity, knowledge-base development and verification, and ends with delivery and a plan for maintaining the knowledge-based system in its application environment. The steps in this three phase process reflect decisions about the context of the knowledge engineering activity as well as decisions concerning the content of the domain and resulting knowledge-base.

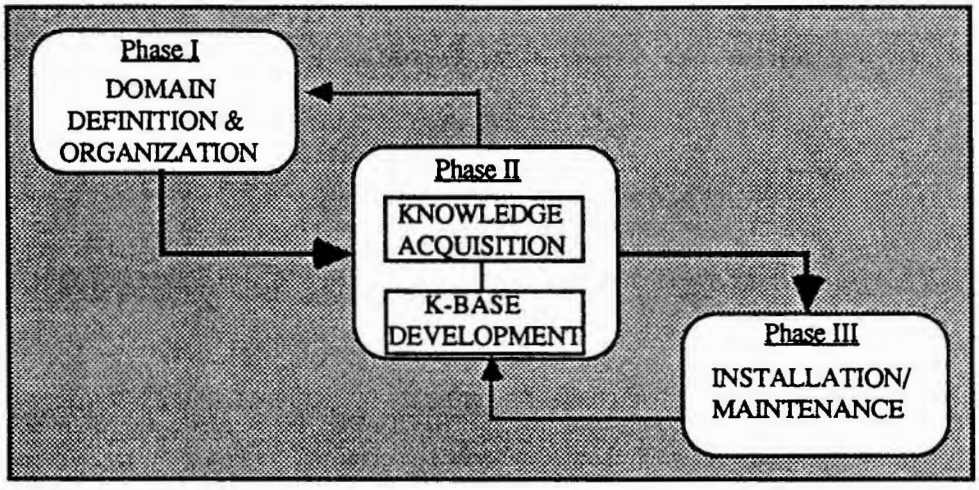


Figure 1. Phases of Knowledge-Base Development

Pragmatically, knowledge acquisition work concentrates on the middle phase. Knowledge acquisition techniques have focused primarily on the direct elicitation of knowledge from a human source in a manner which allows relatively easy representation of this knowledge in a machine environment. The intermediate representation formats used in these knowledge acquisition systems result in knowledge forms which are close-to-machine representable. Little processing is subsequently required to produce a knowledge-base from the output of these knowledge acquisition systems. (Shaw and Gaines, 1987; Abrett and Burstein, 1987; Boose and Bradshaw, 1987; Musen et al, 1987; Gruber and Cohen, 1987; Marcus, 1987). Also, within this phase, work on verification and validation is beginning (Shaw and Woodward, 1988; Stachowitz & Chang, 1988; Hofmeister, 1986; Scambos, 1986).

The predominant activity of this middle phase reflects decisions about knowledge content, organization and structure. Phases one and three, however, reflect issues of context: what application is best suited for a knowledge-based system? Who has responsibility for maintenance? What resources are needed? Domain problems which are cognitive in nature and which demonstrate a logical complexity (rather than common-sense, intuition or hard to define knowledge) are considered appropriate for knowledge-based systems (Martin & Oxman, 1988). Also appropriate are those problems which have a narrow focus and those which have a clearly identifiable pattern or procedure of reaching solutions.

Specifically in phase one, there are important issues about the content of the knowledge and its organization which need attention before phase two activity. Many authors give advice on what kinds of problems are best represented in knowledge-base form.

Knowledge-base designers (eg. Parsaye & Chignell, 1988) suggest a feasibility analysis where issues of domain problem breadth, availability of experts, and stability of procedures in the domain are considered. Others state the need for recognizing the extent of resource commitment in time and funds compared to the eventual return or savings, and the extent to which people would use the system (Martin & Oxman, 1988). Harmon et al (1988) require that the purpose of the knowledge-based system be critically specified before development.

These criteria or guidelines act only as general constraints for domain selection and definition. They do not provide a content analysis of the domain useful for guiding subsequent knowledge engineering activity. Siegel (1986) comes closest to offering a procedure for domain definition. He presents a four-step approach:

1. compartmentalize the domain into sub-problems;
2. define the inference flow;
3. compare client-system dialogues which reflect the conversational format of the proposed system; and
4. define the standards of performance.

However, the procedures for these steps are generally described and do not offer a clear set of testable instructions.

Specific techniques for domain definition and organization are lacking. Techniques which address knowledge content, organization and structure are required to help make critical decisions before concerted and expensive effort in the middle phase of knowledge engineering. Clearly defined techniques based on knowledge acquisition principles aid decision makers in phase one and affect critically the success of latter phase activity. This paper addresses the issue of the development of structured techniques for domain definition and organization. The objectives of this paper are to:

1. describe the content and contextual issues of phase one knowledge-base development activity;
2. discuss briefly how present knowledge acquisition methods are inadequate for domain organization and definition tasks;
3. present a semi-automated tool (Cognosys) which provides a structured methodology for domain definition and organization;
4. outline how Cognosys is used by knowledge-base developers;
5. discuss future developments for Cognosys; and
6. research using the tool.

Domain Organization Issues

Two main issues require consideration: the issue of context and of content in knowledge-base development. The context refers to those decisions concerning the infrastructure or development environment within which the knowledge-base is developed. These include the application parameters, potential users and uses of the knowledge-base, the availability of knowledge sources, the resource constraints of time, funds, and trained knowledge workers, and the political or management climate of deadlines and organizational expectations. These issues define the contextual setting. These decisions may be complex, requiring extensive planning and organization, and can involve many people within the organization. In research settings, these contextual issues are less onerous.

The content refers to those decisions about the knowledge-based system or the domain of interest or application. All knowledge engineering activity includes initial decisions about the domain of application. Partially influenced by the needs of the eventual users and partially by political sensitivities, knowledge-base domain definition is largely and generally defined by the availability of appropriate and interested human knowledge sources and specifically defined by the range of expertise of the knowledge sources. At some point, this domain is defined before the application of specific knowledge acquisition techniques. The definition is usually accomplished by generally focused face-to-face interviews between the knowledge engineer, if available, and the human knowledge source. On this point, descriptions of clearly defined techniques for domain definition are unavailable.

Domain definition and organization is defined as knowledge acquisition activity which incorporates procedures of knowledge elicitation and analysis to produce a domain knowledge model which is used as a basis for further knowledge refinement and structuring. This domain model represents the content and procedural knowledge of a domain at a general level. The finally agreed upon domain model sets the boundaries of the domain. The knowledge models are generated from the problem or application context and the problem demands. They are related to task demands within the domain. These models are used to define the domain and to organize subsequent knowledge engineering activity. Domain organization is a process which is used to determine the scope of the final knowledge-base and the choice of subsequent knowledge acquisition activity. Domain organization is viewed as an essential step in knowledge engineering which structures the domain to isolate and organize key areas of content, then presents knowledge in a manner that "feeds" more specific knowledge acquisition tools.

Knowledge Acquisition Tools and Domain Organization

Currently, all knowledge acquisition procedures assume problem domain definition in a manner which is conducive to starting knowledge engineering activity. The application of knowledge acquisition techniques further assumes that initial decisions have been made about the boundaries of the domain: what is included and what is not. These assumptions are demonstrated by the very few detailed descriptions of this phase one activity. In descriptions of knowledge acquisition techniques, initial domain definition is implied by the fact that the processes and techniques begin with a clearly defined domain and at a level of detail which suggests initial organization. It is assumed that an interview or a series of interviews has been completed, and that the guidelines for domain selection were followed (Waterman, 1986).

Automated knowledge acquisition systems such as KSSO (Gaines, 1987) and AQUINAS (Boose and Bradshaw, 1986), require as input a series of conceptual primitives called elements or entities which are then organized along dimensions or constructs for further processing and analyses. These tools are extremely powerful but they assume a certain level of specificity or granularity of knowledge. Once the purpose of the elicitation has been defined, the generation of cognitive primitives is easily forthcoming. The purpose itself results from a content analysis of the domain.

Other automated systems require the input of object-attribute pairs which are then organized into roles to determine domain knowledge (KNACK: Klinker et al, 1987), or ask the expert for lists of events such as hypotheses and evidence which are then associated (MOLE: Eshelman et al, 1987), or require the expert to slot-in a variety of information types in the form of events, objects, names, formulae (SALT: Marcus, 1987).

Common to these knowledge acquisition tools is the identification of specific attributes of cognitive primitives used as input. They were not built with phase one activity in mind, and their level of granularity is not suited to phase one content analysis. Each tool requires that domain definition and organization be completed prior to use of the tool or procedure. Also, these tools attempt to provide close-to-codeable output so they are extremely rigorous in their knowledge representation schemes. They have proven very effective in uncovering, organizing and representing patterns of knowledge. These qualities are essential for phase two activity but not for phase one.

Principles of Domain Definition and Organization

The present knowledge acquisition tools, though not suitable for phase one domain definition, offer clear guidelines and rigorous standards for the development of domain organization techniques. Domain organization methodology falls into the category of Knowledge Support, in that it aids knowledge engineering activity and supports and facilitates human knowledge processes. Shaw and Gaines (1987) have outlined the general design requirements for knowledge support systems. Domain organization methodology has followed these guidelines. Systems and procedures which incorporate domain organization methodology need to:

1. be domain independent;
2. access a diversity of knowledge sources including text, interviews, and observations;
3. encompass a diversity of perspectives including partial or contradictory input from human knowledge sources;
4. be directly applicable by a human knowledge source without intermediaries;
5. present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations;
6. encompass a diversity of forms of knowledge and relationships between these forms;
7. allow users to apply knowledge in a variety of domains and to freely experiment with the implications;
8. make provision for validation and verification; and
9. be grounded on a well-developed explicit theory.

Although not speaking specifically about knowledge support methodology, Garg-Janarden and Salvendy (1987) support many of the above design principles. They say that any methodology must have a theoretical framework which helps to determine the types of knowledge for consideration and representation. These authors also suggest that any methodology should be capable of representing underlying knowledge structures which are not obvious in the surface analysis, but which are present upon a deeper analysis in the form of deep, causal and strategic knowledge. Quantitative methods should also be used wherever possible as a basis of analytic procedures. Finally, the methodology should provide procedures for examining and modifying the resultant knowledge structures. Gruber et al (1986) summarize another constraint when they suggest that knowledge acquisition techniques allow human knowledge sources to set their own level of discourse. Such systems do not demand that users organize their knowledge to fit the input requirements, but rather the reverse.

More recently, work in recognizing and defining the network of task demands within a domain has suggested another set of guidelines for domain organization methodology. McDermott (1988) states that a clear understanding of task requirements is essential for knowledge acquisition activity - that task demands structure, order and define subsequent knowledge representation and knowledge processing decisions. Chandrasekaran and Bylander (1988) have identified generic tasks which are defined by their own characteristic requirements for knowledge representation and knowledge processing, and which can act as modules for building knowledge-based systems. It is essential that domain organization methodology incorporate an understanding of task demands.

COGNOSYS

In its present form, Cognosys is best described as a knowledge engineer's tool for structuring initial discussions with a human knowledge source, and for constructing domain models. It operates in a semi-automated fashion. Some procedures of Cognosys require informed human judgment and manipulation of the knowledge. Cognosys offers a convenient format for organizing knowledge. For other procedures, Cognosys performs analyses and manipulations for the knowledge engineer. The prototype of Cognosys is in Hypertalk and runs on the Mac II or SE. The purpose of Cognosys is to aid the knowledge engineer in initial domain organization by helping to organize knowledge, in a general domain model, from the human knowledge source and to generate a useful representation of this model.

Input to Cognosys

The beginning point in the use of Cognosys is a set of written descriptions of the domain at a level which gives a complete overview of the domain. The human knowledge source is asked to generate and/or provide:

1. a descriptive statement of the overall domain;
2. a list of general objectives which must be met in order to successfully solve problems or make decisions within the domain; and
3. a collection of short paragraphs which are descriptive of the tasks required to meet each of the objectives.

The last may be from the individual's own work or from articles referred to by the individual and specifically chosen for this purpose. These three types of description act as the raw input for the knowledge engineer and Cognosys. Once this information is provided, Cognosys and the knowledge engineer cycle through the process outlined in Figure 2 . Initial discussions result in the generation of material which is processed by the

knowledge engineer and Cognosys, resulting in a model which is used as a basis for discussion between the knowledge engineer and the human knowledge source. These discussions lead to further refinement of the domain model until the human knowledge source is satisfied that it represents the emphasized domain components. The final model represents the content of the domain and the various objectives and task demands required.

Overview of Cognosys

Figure 2 outlines the three main sub-components of the Cognosys program: Analysis, Synthesis, and Support. The first two are discussed in detail, the third only generally. The knowledge engineer follows the procedural steps outlined by Cognosys to organize knowledge from initial discussions with the human knowledge source. The Analysis component on the left results in the partitioning and sub-partitioning of the knowledge and works top-down. The Synthesis path in the middle of Figure 2 starts at the bottom with knowledge units or K-units and moves upward to build a domain model. The Support component houses help information, a map of the system, and note records and storage.

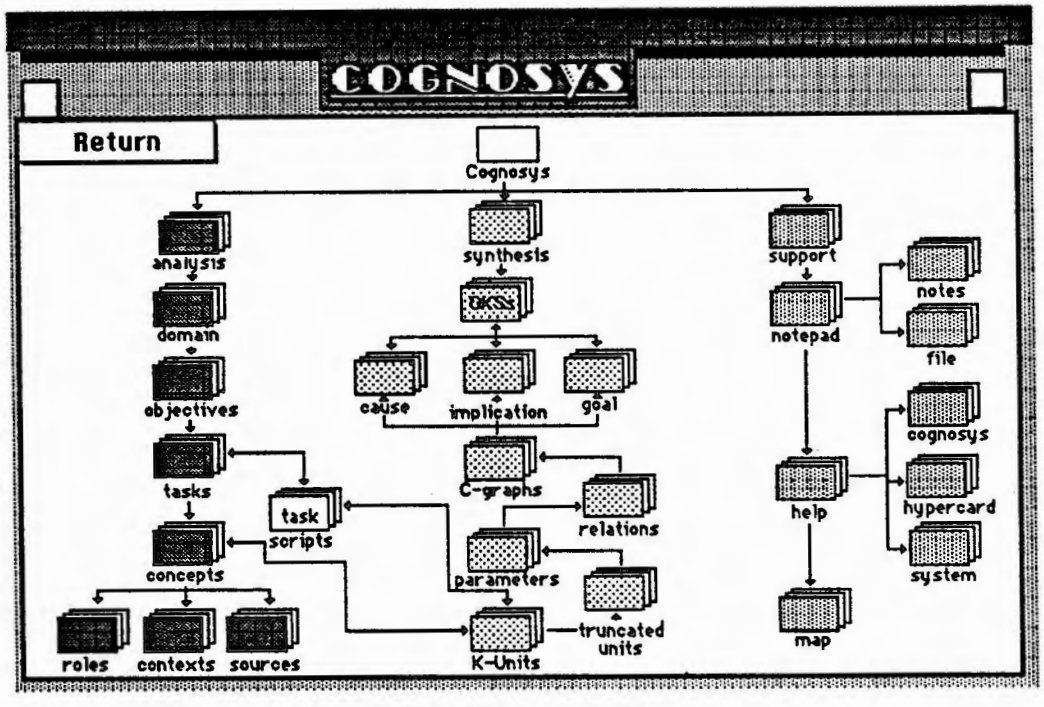


Figure 2. Map of the COGNOSYS Program

Cognosys screens correspond to the 3 sub-components of the program in that they offer different choices. The most pervasive and predominant feature is the "navigational cube" found always in the left hand corner of the screen (Figure 3). This device is used to

navigate the Cognosys program. When required, another, smaller cube appears. The cubes act as indicators of the current procedure as well as active instruments of movement to other procedures.

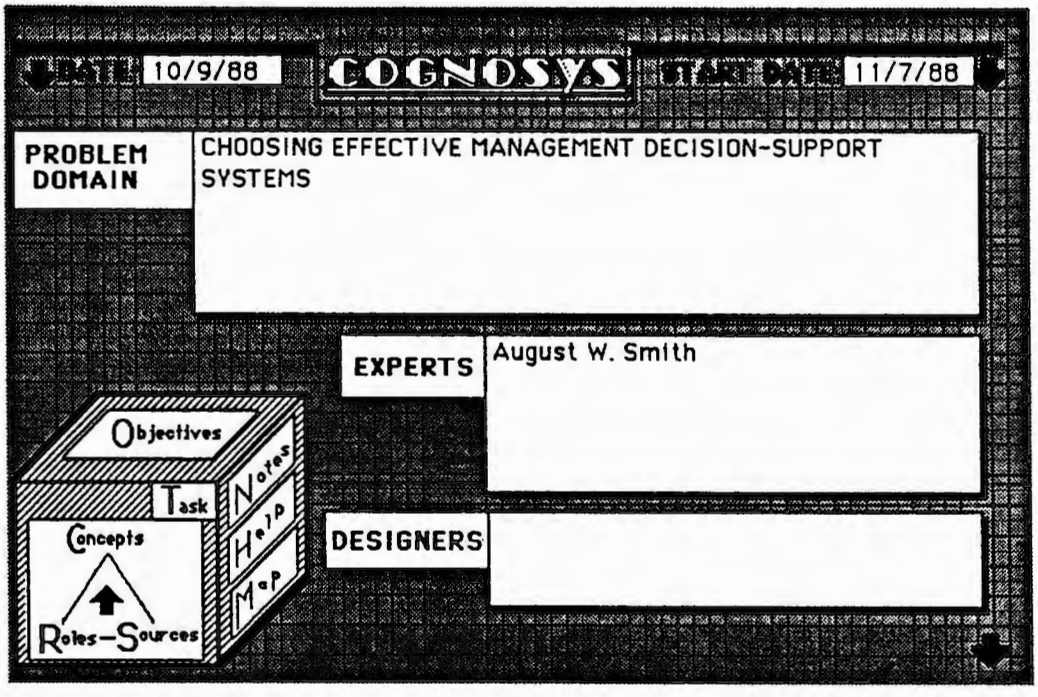


Figure 3. Navigational Cube and Domain Title

Analysis

The purpose of this component is to:

1. organize the domain into objectives (general goals);
2. analyze the objectives into tasks;
3. classify the task types;
4. analyze the tasks into functional and structural concepts;
5. classify the role of the concepts;
6. organize the source of the concepts; and
7. record any attributes of the concept.

Although not all of the information generated in steps one to seven is used in the Synthesis component, this information is necessary for two reasons. The first is to aid in search and selection routines for the cataloging of information: the second to act as potential "feed-stock" for more specific knowledge acquisition tools used later in the knowledge engineering process.

1. The Objectives Level

Figure 4 shows the screen for the first objective which has been entered manually or from file. A short title for reference is also entered. Other objectives are added until all are entered. The task level is then chosen for each objective in turn.

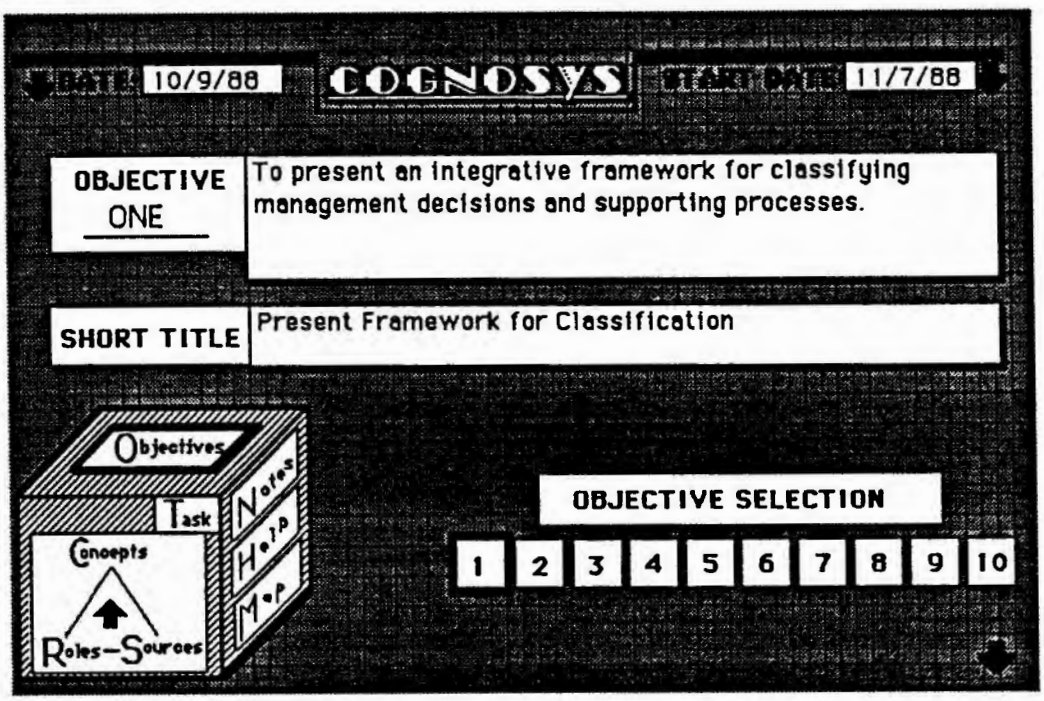


Figure 4. The "Objectives" Level

2. The Task Level

Figure 5 shows a task description for a given objective. This is a short paragraph containing 5 to 6 sentences. These descriptions are entered manually or from file. Now the task is typed using Greeno's taxonomy of task types. The knowledge engineer reviews each task description and identifies the various types: pattern finding, transformation and arrangement, or any combination of the three. Pattern-finding tasks require the decision-maker to identify a known pattern of objects, data or ideas. Transformation tasks are characterized by a given initial pattern and a desirable end pattern. The problem-solver is required to use legitimate steps to transform the initial pattern to coincide with the desired pattern. Arrangement tasks refer to those tasks for which only pieces of information are given and a final arrangement or pattern is required to meet certain design or functional criteria. Often, tasks can have more than one 'type' assignment. If this occurs, all types are recorded and the task may be further analyzed to a single type by the human knowledge

source in the next iteration of Cognosys.

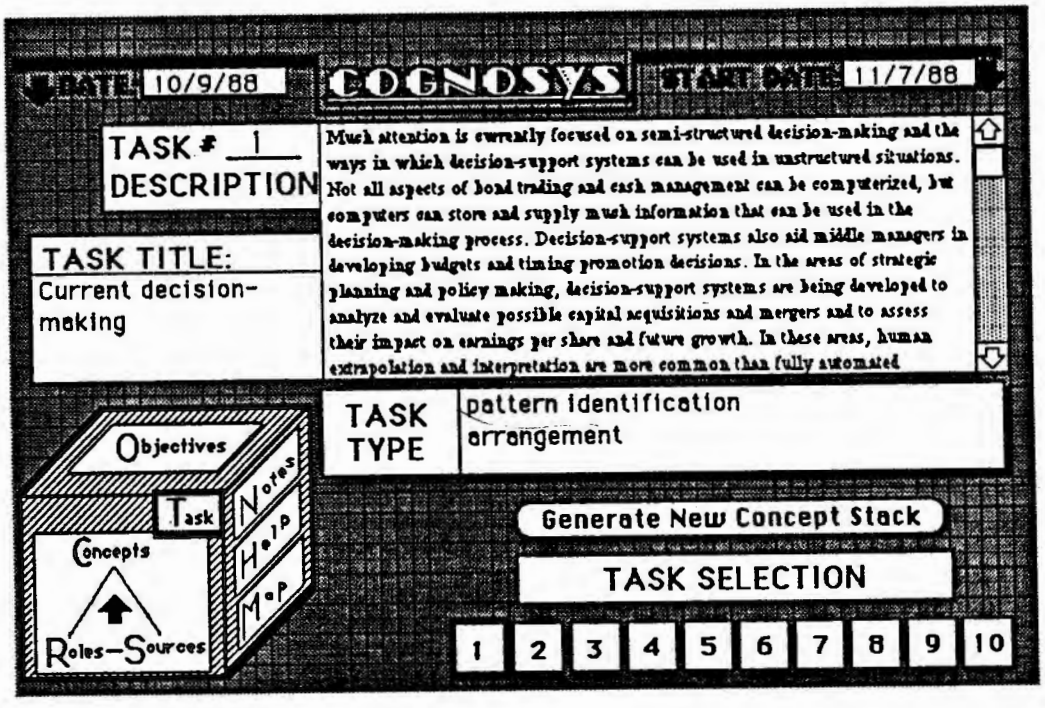


Figure 5. The "Task" Level

At this level of analysis, the Greeno categories provide an appropriate format for task typing. Chandrasekaran (1988) and McDermott (1988) have developed task-level categories for describing task demands, but these systems were developed for analysis at a more specific level of knowledge acquisition. They are extremely rigorous and useful frameworks but are not required at the domain organization level of analysis.

3. The Concept Level

This level of analysis reflects the linguistic assumption that a word (noun, adjective, verb or adverb) is a single concept (Graesser, 1985; Fauconnier, 1987). The concepts are selected from the task description paragraph by the knowledge engineer and classified as either structural (noun or adjective) or functional (verb or adverb) (Figure 6). These single word concepts act as cognitive primitives for the analysis sub-component of Cognosys. The single concepts are stored together with their task context and any attributes which may have been denoted in the task paragraphs (Figure 7).

The source of information concerning the concept is also recorded. The source could be the human knowledge source, calculated information or a data source. Finally, the role of

the concept is declared. This declaration may be a descriptor of a structural or functional concept, but more often, the best way to describe the role is to place the concept in the phrase within which it appears in the paragraph.

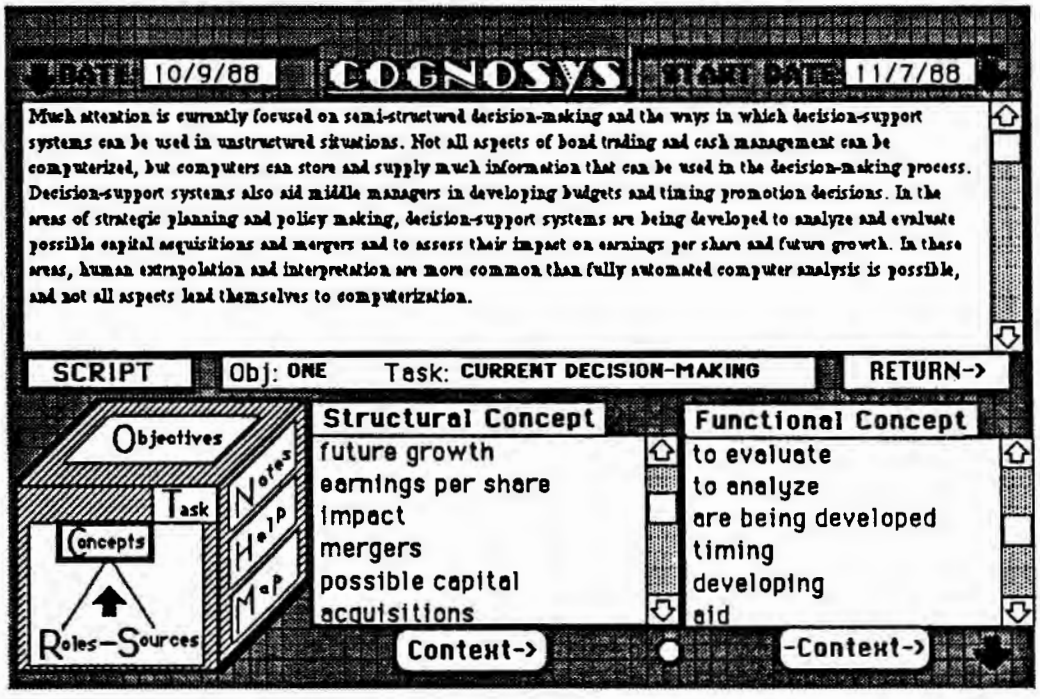


Figure 6.Classification of Concepts

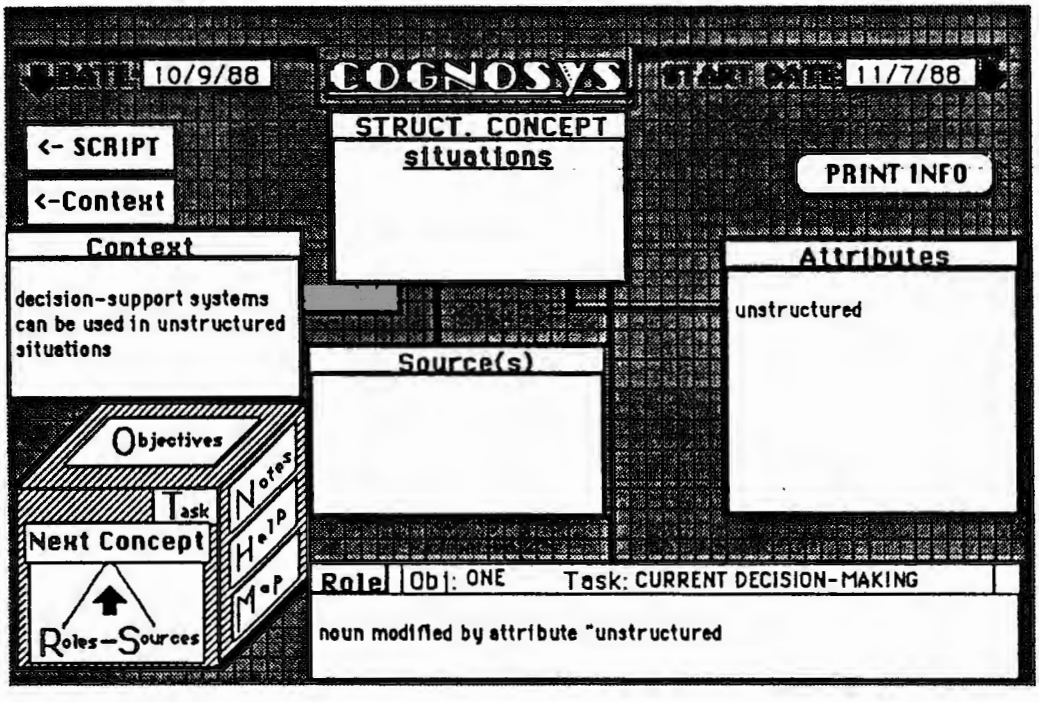


Figure 7. The "Concept" Level

Often, the first run through the Cognosys program does not provide enough information for the completion of each category for the concept description (Figure 7). This information is usually forthcoming in subsequent discussions.

4. Summary

This portion of the Cognosys program provides the knowledge engineer with a methodology for analyzing the problem domain into primary objectives and each subsequent set of tasks. The knowledge engineer requests of the human knowledge source a breakdown of the domain into required operational or procedural objectives required in order to arrive at a satisfactory solution to classes of domain problems. These objectives are then decomposed into tasks with short descriptive paragraphs. The tasks are typed and the concepts isolated, typed, and stored with their sources and attributes.

Synthesis

Most of the organization of the domain model occurs as a result of synthesis activity. The purpose of this Cognosys component is to:

1. isolate and type discrete knowledge primitives (K-Units);
2. identify relationships among K-Units;
3. display K-Unit relations for each task in conceptual graphs to form a Specific Knowledge Structure (SKS);
4. combine task-level conceptual graphs into General Knowledge Structures (GKS) for each objective; and
5. combine each GKS for objectives into a domain model.

1. Methodology

The method used to develop the GKSs is based on work by Graesser (1981) and Graesser and Clark (1985). This method of determining structures and procedures of implicit knowledge has been successful in understanding how readers comprehend written text. The primary component is that of a GKS which is described as "a rich configuration of information which contains a combination of sub-structures..[which are considered].. declarative knowledge structures" (p.4). Other examples of general knowledge formats or packages are frames (Minsky, 1975), scripts and/or plans (Schank and Abelson, 1977), productions (Anderson, 1983), memory organization packages (Schank, 1982), and thematic abstract units (Winston, 1984). These systems each provide a framework for understanding based on syntactical analysis. Graesser's GKSs concentrate rather on

mapping the content and structure used in understanding specific text. This approach was selected because it was more content-oriented than the others and because it provided a framework for modelling " the actual content and organization of knowledge in the cognitive system" (p. 47) of the human knowledge source. Also, it was chosen because of its representational quality. The representational system for GKSs is the conceptual graph. With variations, this type of system has been used successfully by many researchers; most notably Sowa (1984). The conceptual graphs used in Cognosys are, at times, more restricted than those of Sowa in that they represent a set of global relationships between and among concepts and therefore cover fewer examples or types of precise relationships. At other times, the conceptual graphs are more flexible and encompassing because they do not require the rigor of Sowa's conceptual graphs since his goal is to describe his conceptual graphs in close-to-codeable form. It is not unlikely, however, that many of Sowa's inference techniques can be used with Graesser-type conceptual graphs (C-graphs).

The C-graphs represent knowledge at two levels: the specific and the general (or the task and the objective). Specific knowledge structures (SKSs) represent task level knowledge. These are combined to form a GKS for each objective in the domain. Each consists "of a set of categorized statement nodes which are interrelated by a network of directed, relational arcs" (Graesser and Clark, 1985, p. 48). The statement node is similar to other knowledge representations formats such as Sowa's conceptual graphs and Anderson's productions. These nodes are categorized as:

1. a *state* which represents a situation remaining unchanged throughout a period of time;
2. an *event* which represents a situation which changes within a period of time;
3. a *goal* which describes a state or event which is desired; or
4. a *style* which modifies states or events by specifying the manner in which an event occurs or further distinguishes a state.

Relational arcs act to connect the nodes within a restricted framework, but they capture relationships which frequently exist between nodes in a knowledge structure. The types are:

1. *Consequent* arcs which demonstrate how one node can causally lead to another node, enable another node to happen, or show how one node results in the occurrence of another node (temporally bound);
2. *Implies* arcs which are weaker than consequent arcs but are also used to relate event or state nodes when both node types occur at the same time (temporally free);
3. *Reason* arcs which are used to denote sequences of goal state nodes such that one node is "a reason for another node";

4. *Outcome* arcs which specify whether goal nodes are achieved (temporally bound);
5. *Initiates* arcs which represent relations between state/event nodes which invoke goal nodes (temporally bound);
6. *Manner* arcs which specify how an event node is completed;
7. *Property* arcs which further define state nodes by style and/or context (temporally free); and
8. *Set Membership* arcs which reflect the set inclusion or class of inclusion of nodes.

The combinations of nodes and arcs are represented in C-graph form which can then undergo a variety of analyses and can be combined for greater complexity.

2. K-Units and their Relationships

The knowledge engineer uses the same task paragraphs as in the Analysis component of Cognosys. In this procedure, however, complete sentences are extracted from the paragraph and then truncated into noun-verb-noun units and typed (state, event, goal, style) (Figure 8). The result is a list of K-Units and their respective types.

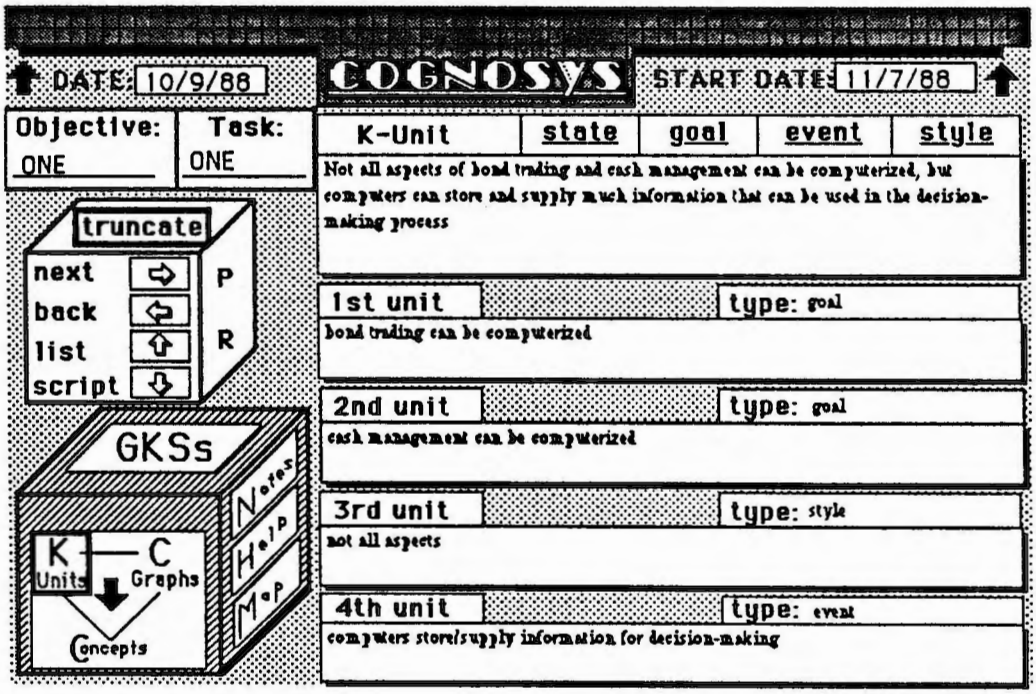


Figure 8. Truncating K-Units from Sentences

Cognosys then displays each combination of two K-Units so that the knowledge engineer can assign a relationship arc (Figure 9). This is done with reference to the task

paragraph (see Figure 7). The result of this activity is a display of K-Units and their inter-relationships.

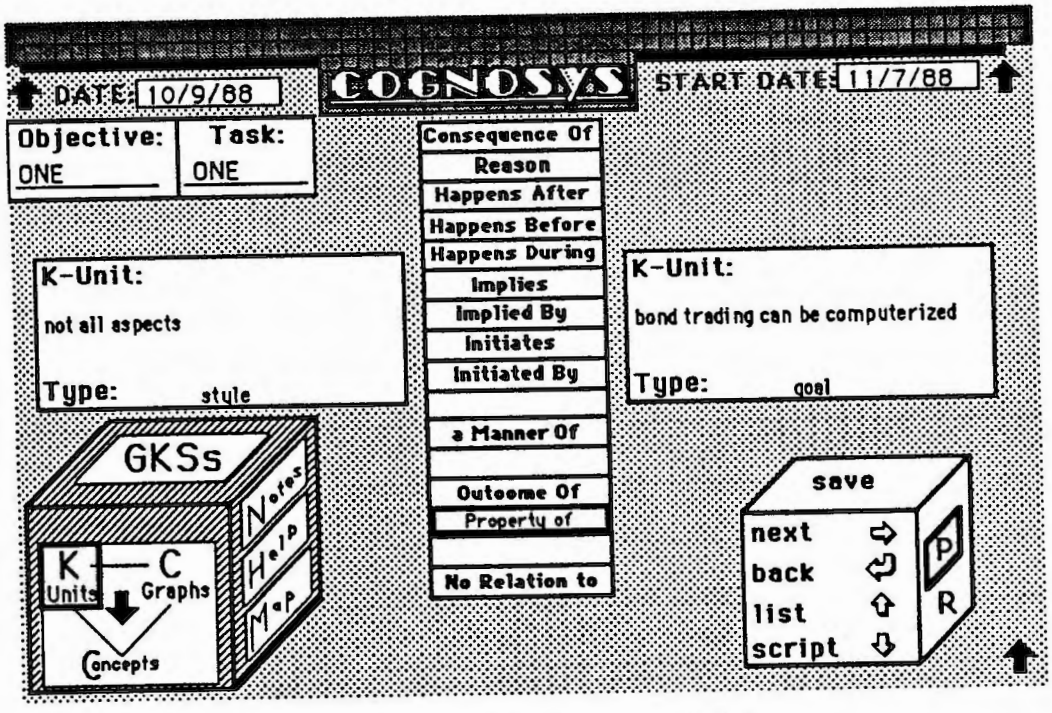


Figure 9. Assigning Relations Between K-Units

3. C-graphs

The representation of nodes and arcs in C-graph form requires analysis by Cognosys and direct arrangement by the knowledge engineer. Structural Centrality is defined as " the number of arcs that are directly connected to a given node" (Graesser and Clark, 1985, p.102) and is used to guide arrangement. Cognosys uses the list of K-Unit relations to calculate the structural centrality score for each node. This score is displayed with each K-Unit and is used by the knowledge engineer in C-graph construction. Those with higher scores are placed closer to the centre of the graph. Each C-graph represents one task. Figure 10 shows a C-graph for the example task as arranged by a knowledge engineer. This C-graph represents a specific knowledge structure (SKS) for the task.

The knowledge engineer completes this procedure for each task description for each objective. When the procedure is complete, the knowledge engineer has a series of C-graphs which can be combined to represent the objective or general knowledge structure (GKS). Because of its present hypercard environment, Cognosys cannot display a complete GKS in C-graph form. These can be constructed from the print-outs of each SKS C-graphs. Figure 11 shows a completed GKS. The nodes and arcs for the tasks are

shown. In this example, some task SKSs overlap indicating that similar or the same concepts are used for different tasks demands.

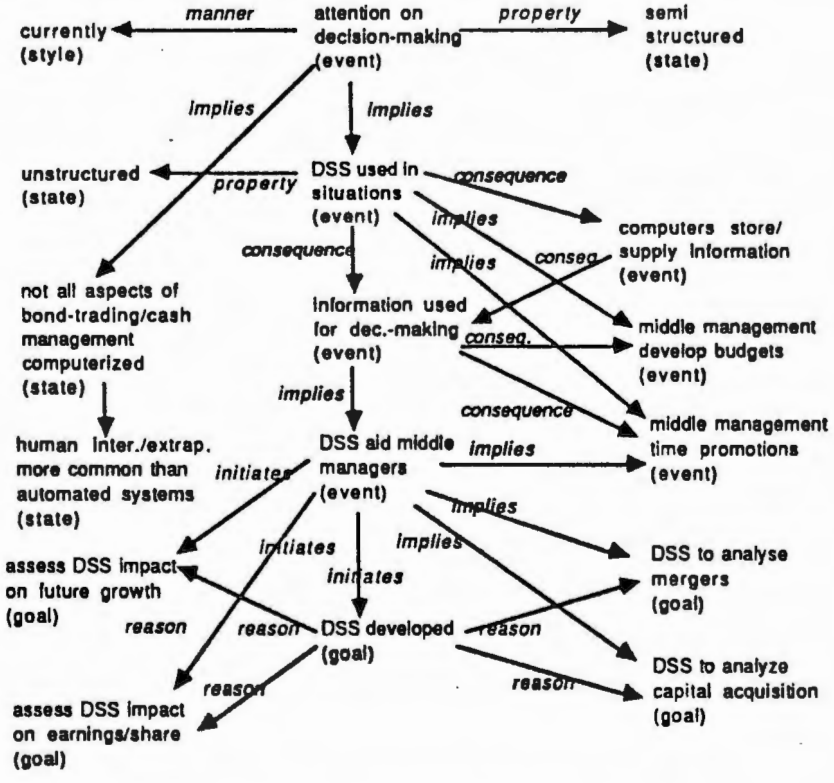


Figure 10. Sample C-graph for Task Example

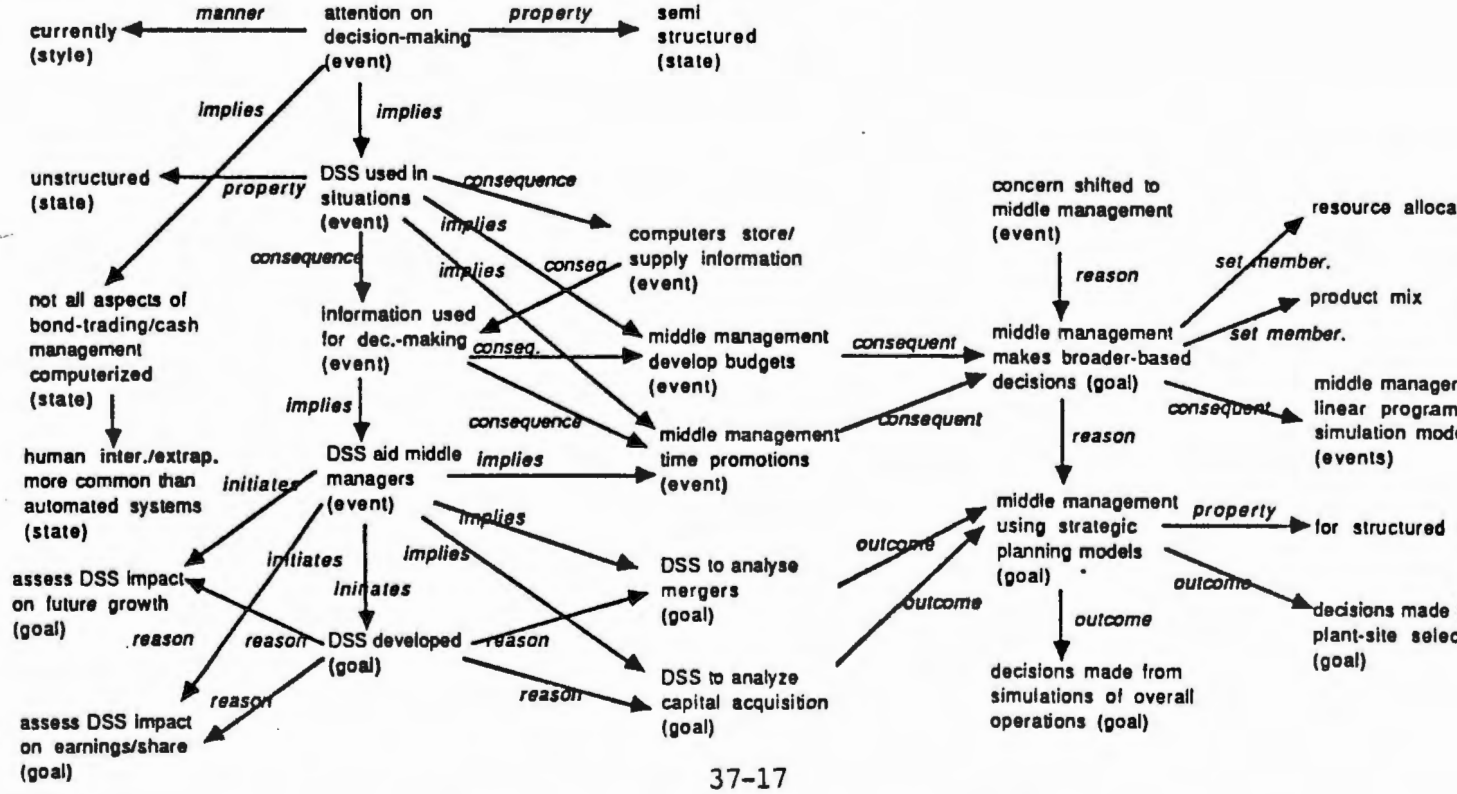


Figure 11. General Knowledge Structure (GKS) in C-graph Representation

4. Operations on GKSs

Cognosys is able to use the GKS to isolate useful sub-structures. Cause-effect, implicational, goal-oriented, and taxonomic sub-structures are useful for understanding the human knowledge source's knowledge structures for the domain. Goal-oriented sub-structures represent organized plans and activities which animate agents use to reach a set of goals. These goal structures are generated by selecting all reason, outcome, initiates and manner arcs. Sequences of events and states are graphed to display the path to goals in a temporally bound manner. These sub-structures (see Figure 12) are also useful in further discussion with the human knowledge source because they allow clarification and elaboration of temporal relationships, proper event and state sequences and goal specification.

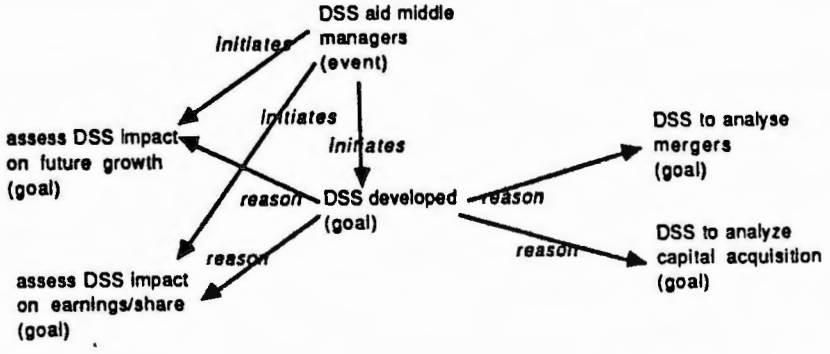


Figure 12. Goal-Oriented Sub-structure

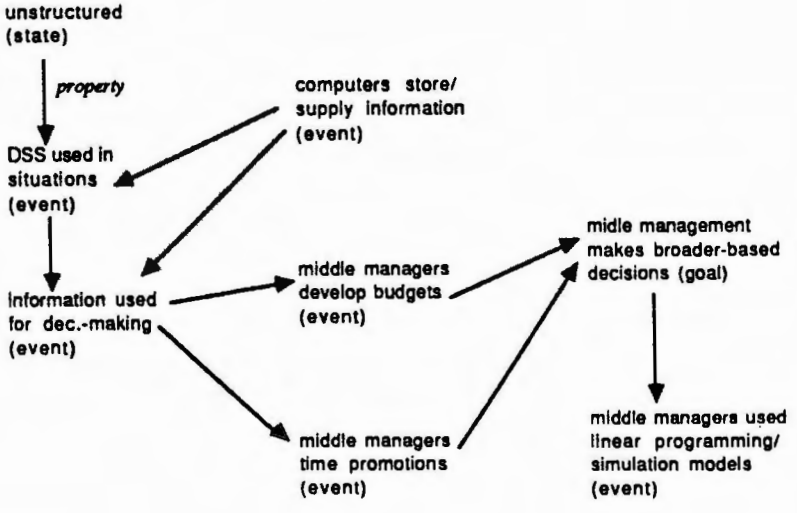


Figure 13. Cause-Effect Sub-Structure (all Consequence Arcs)

Cause-effect sub-structures capture what are specifically considered as event-effect relationships. These structures are not governed by the goal structures but represent action-initiated activity and are thus refinements of goal structures. They are generated by selecting all consequent arcs. They result in event-event node sequences which can be embellished to include related goals, states and manner nodes for further explanation (Figure 13). These sub-structures of GKSs are useful in understanding the human knowledge source's attribution of cause in the domain. This can be further elaborated in subsequent discussions.

Implicational sub-structures reflect the existence of deductive or inductive reasoning portrayed in the GKS. They are generated by selecting all implies arcs (Figure 14). These sub-structures are useful in further discussions because they allow elaboration and explanation of these inferences.

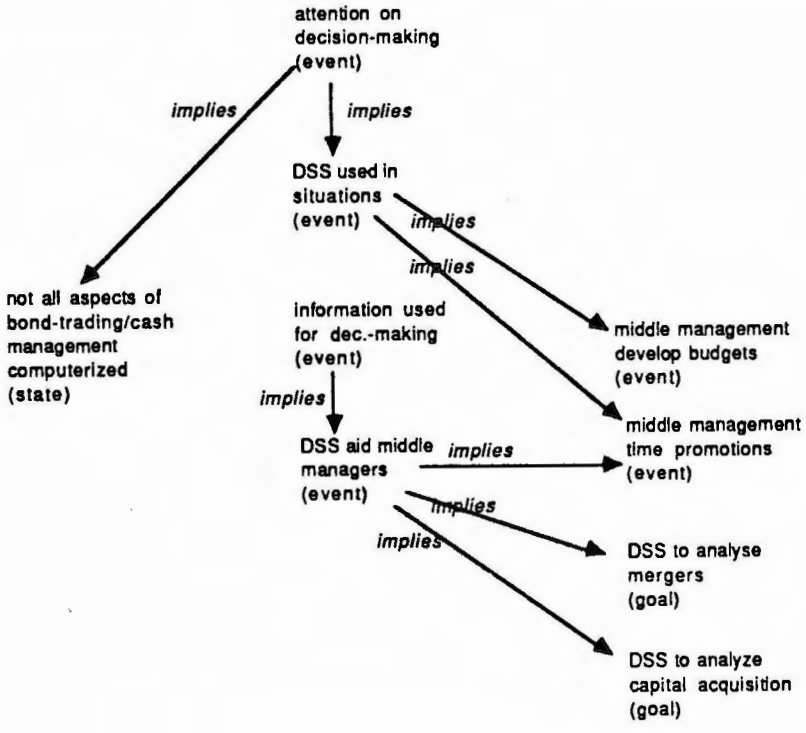


Figure 14. Implicational Sub-Structure

Finally, taxonomic sub-structures represent those relationships which suggest set and sub-set membership. These sub-structures are generated by selecting the property and set-membership arcs (see Figure 15), and are used to identify hierarchical structures within the domain.

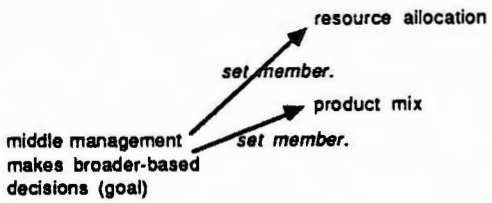


Figure 15. Taxonomic Sub-Structure

5. Summary

The knowledge engineer applies set of procedures to the task paragraphs which result in the formation of K-Units. These act as the primitives which are then typed and combined through relational arcs to form specific knowledge structures: one for each task. These SKSs are represented in C-graph form. The SKSs are then combined to form a GKS for the set of objectives. The combination of GKSs defines the domain.

Domain Definition and Organization

The problem domain is too complex for the human knowledge source, let alone the knowledge engineer, to elucidate without aid. Both individuals can attend to only parts of the domain at any one time, so they require a full model of the domain to aid in discussions about the domain and its components. Cognosys helps the knowledge engineer construct this domain model with very little time required from the human knowledge source. Through a series of clarification interviews, the knowledge engineer re-designs and tests domain models prior to expensive time in specific knowledge elicitation. Apart from clarifying nodes and arcs, portions of the model are culled or expanded as required. The boundaries of the domain are clearly identified, the central domain content and structures defined and the edges of expertise recognized. Cognosys acts as a facilitation tool in these initial design stages.

Now, decisions are made about the next steps in the knowledge-base system development. Several knowledge-based systems may be identified. The human knowledge source identifies a specific problem within the domain to focus on, or prefers to deal with one area only within the defined domain model. Decisions for resource commitment are made based on the domain model. For the knowledge engineer, general task types are identified so decisions about fine-grained knowledge acquisition tools, knowledge representation formats, and inference processes are made. Sources of domain knowledge are identified and organized.

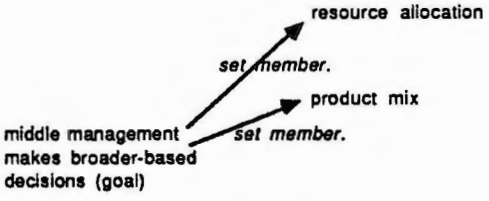


Figure 15. Taxonomic Sub-Structure

5. Summary

The knowledge engineer applies set of procedures to the task paragraphs which result in the formation of K-Units. These act as the primitives which are then typed and combined through relational arcs to form specific knowledge structures: one for each task. These SKSs are represented in C-graph form. The SKSs are then combined to form a GKS for the set of objectives. The combination of GKSs defines the domain.

Domain Definition and Organization

The problem domain is too complex for the human knowledge source, let alone the knowledge engineer, to elucidate without aid. Both individuals can attend to only parts of the domain at any one time, so they require a full model of the domain to aid in discussions about the domain and its components. Cognosys helps the knowledge engineer construct this domain model with very little time required from the human knowledge source. Through a series of clarification interviews, the knowledge engineer re-designs and tests domain models prior to expensive time in specific knowledge elicitation. Apart from clarifying nodes and arcs, portions of the model are culled or expanded as required. The boundaries of the domain are clearly identified, the central domain content and structures defined and the edges of expertise recognized. Cognosys acts as a facilitation tool in these initial design stages.

Now, decisions are made about the next steps in the knowledge-base system development. Several knowledge-based systems may be identified. The human knowledge source identifies a specific problem within the domain to focus on, or prefers to deal with one area only within the defined domain model. Decisions for resource commitment are made based on the domain model. For the knowledge engineer, general task types are identified so decisions about fine-grained knowledge acquisition tools, knowledge representation formats, and inference processes are made. Sources of domain knowledge are identified and organized.

Cognosys concepts provide a feed-stock for specific knowledge acquisition tools. For example, KSS0 requires "entities" which act as the basis of elicitations and can then be used in knowledge-base development. These elicitations are most effective and accurate when the "entities" (or concepts in Cognosys) are at the same level of generality or granularity, are ordered heterarchically, rather than hierarchically, and share general commonality or set relationship. Further refinement of concepts into entities is accomplished through a series of node ratings on three dimensions. The first is typicality which requires the human knowledge source to rate all nodes on how typical they are of the GKS. Given the GKS, how frequently would the node occur. The second is necessity, which refers to the question of how necessary a node is for the GKS to occur. In other words, would the GKS exist/occur if the node were absent? The last scale is distinctiveness, which refers to the degree to which the node is particular to a given GKS or if it appears in many GKSs with in the defined domain. Low distinctiveness scores indicate a common activity, goal or state within the domain.

Future Developments for Cognosys

As a semi-automated knowledge acquisition tool, Cognosys requires informed judgment from the knowledge engineer. Whether the knowledge engineer role is filled by a domain expert or not, using Cognosys requires knowledge about task analysis and GKS development. Successive versions of Cognosys will contain increasingly more automated procedures. The present hypercard environment has been useful for rapid development of the prototype but a more powerful object-oriented environment such as smalltalk is better suited to more automation. Specifically, automatic generation of C-graphs for SKSs and GKSs is necessary. Also, multiple windows would make some processes and decisions easier.

A wide variety of analyses are possible using GKSs: using node and arc types and combinations to identify clusters and sub-clusters of node/arc occurrence, procedures for GKS comparisons, and for identifying implicit knowledge structures. These are possible within the hypercard environment but are much more easily implemented in smalltalk.

Also, additional support processes would make Cognosys a more effective and general tool (Figure 16). For example, a knowledge-base of GKSs which allowed filing, sorting, matching, and combination retrieval would benefit the user. A library containing references of knowledge engineering tools and procedures, their purpose, and their application is useful. A knowledge-base for advising Cognosys users on the system itself and for selecting knowledge acquisition tools is possible. Finally, output formats for a wide variety of specific knowledge acquisition tools is necessary.

Cognosys as a Research Tool

Cognosys provides a useful framework for research into human knowledge structures and the usefulness of these in knowledge-base design and development. Research areas include:

1. identifying biases in GKSs for different task demands and human knowledge sources;
2. comparing GKSs over time to track knowledge development;
3. identifying the influence of human knowledge source cognitive style variables on knowledge acquisition;
4. applying Cognosys procedures to finer-grained knowledge acquisition activity; and
5. identifying GKS concomitants to generic level tasks (Chandrasekaran, 1988) and task methods (McDermott, 1988).

References

Abrett, G. & Burstein, M.H. The KREME knowledge editing environment. **International Journal of Man-Machine Studies**, Vol 27(2), 103-126.

Anderson, J.R. (1983) **The Architecture of Cognition**. Cambridge, Mass: Harvard University Press.

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 of Man-Machine Studies**,Vol. 26 (1), 3-28.

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**,Vol. 26 (2), 231-244.

Chandrasekaran, B. & Bylander, T.(1988) Task-specific problem-solving architectures. **Tutorial-AAAI**, St. Paul. Morgan-Kaufmann.

Eshelman, L., Ehret, D., McDermott, J. and Tan, M. (1987) MOLE: a tenacious knowledge acquisition tool. **International Journal of Man-Machine Studies**,Vol. 26 (1), 41-54.

Fauconnier, G. (1985) **Mental Spaces: Aspects of Meaning Construction in Natural Language**.Cambridge, Mass.: MIT Press.

Gaines, B.R. (1987) Advanced expert system support environments. **Proceedings of the Second Knowledge Acquisition for Knowledge-Based Systems Workshop**. Banff, Canada.

Garg-Janardan, & Salvendy,G. (1987) A conceptual framework for knowledge-elicitation. **International Journal of Man-Machine Studies**, Vol 26(4), 521-532.

Graesser, A.C. & Clark, L.F. (1985) **Structures and Procedures of Implicit Knowledge**. Norwood, NJ.: Ablex Publishing.

Graesser, A.C. **Prose Comprehension Beyond the Word**. New York: Springer-Verlag.

Greeno, J.G. & Simon, H.A.(1986) Problem solving and reasoning, In (Eds.), R.C. Atkinson, R.J. Hernstein, G. Lindsey & R.D. Luce, **Steven's Handbook of Experimental Psychology**. New York, NY: Wiley.

Gruber, T.R. & Cohen, P.R. (1987). Design for acquisition: principles of knowledge system design to facilitate knowledge acquisition. **International Journal of Man-Machine Studies**, Vol 26(2), 143-160.

Harmon, P, Maus, R. & Morrissey, W. (1988) **Expert Systems: Tools and Applications**. Toronto: Wiley.

Hofmeister, A.M. (1986) Formative evaluation in the development and validation of expert systems in education. **Computational Intelligence**, 2, 65-67.

Marcus, S. (1987) Taking backtracking with a grain of SALT. **International Journal of Man-Machine Studies**, Vol. 26 (4),383-398.

Martin, J. & Oxman, S. (1988) **Building Expert Systems: A Tutorial**. Englewood Cliffs, NJ.: Prentice Hall:

McDermott, J. (1988) Exploiting task structure to automate knowledge acquisition. **Invited Address, AAAI**, St. Paul.

Minsky, M. (1975) A framework for representing knowledge. In P.H. Winston (Ed.), **The Psychology of Computer Vision**. New York: McGraw Hill.

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**, Vol 26(1), 105-121.

Parsaye, K. & Chignell, M. (1988) **Expert Systems for Experts.** Wiley: Toronto.

Scambos, E.T. (1986) A scenario-based test tool for examining expert systems. **IEEE**, 131-135.

Schank, R.C. & Abelson, R. (1977) **Scripts, Plans, Goals, and Understanding.** Hillsdale, NJ: Erlbaum.

Schank, R.C. (1982) Reminding and memory organization: an introduction to MOPs. In W.G. Lehnert & M.H. Ringle (Eds), **Strategies of Natural Language Comprehension.** Hillsdale, NJ: Erlbaum.

Shaw, M.L.G, and Gaines, B.R. (1987) KITTEN: Knowledge initiation and transfer tools for experts and novices. **International Journal of Man-Machine Studies**. Vol 27(3), 251-280.

Shaw, M.L.G. & Woodward, J.B. (in press). Validation in a knowledge support system: replication and consistency with multiple experts. **International Journal of Man-Machine Studies.**

Siegel, P. (1986) **Expert Systems: A Non-programmers Guide to Development and Applications**. Blue-Ridge Summit, PA: TAB.

Sowa, J.F. (1984) **Conceptual Structures: Information Processing in Mind and Machine.** Menlo Park, California: Addison-Wesley.

Stachowitz, R.A. and Chang, C.L. (1988) Verification and Validation of expert systems. **Tutorial-AAAI**, St. Paul: Morgan-Kaufmann: San Mateo, California.

Waterman, D.A. (1986) **A Guide to Expert Systems**. Don Mills, Ontario: Addison-Wesley.

Winston, P.H. (1984) **Artificial Intelligence (2nd ed)**. Reading. Prentice-Hall.

3rd AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, November, 1988

PRED: An Interface Development Tool for Editing Primitives in Knowledge Representation Languages*

Shun-en Xie, David F. Dumaresq,
Philip H. Winne
Instructional Psychology Research Group, Faculty of Education
Simon Fraser University, Burnaby, B.C. V5A 1S6, Canada
email: cc.sfu.ca!David_Dumaresq@ubc-cs.uucp

ABSTRACT

PRED allows domain experts to easily create frame-based windowed knowledge acquisition interfaces for knowledge systems development. These interfaces provide visibility for complex domain knowledge and speed up the entry of knowledge by allowing the specialist to supply lists of default values. Specialists first use PRED to schematize a knowledge unit, building an internal model. The internal model becomes a map to create a panel or the data structure to contain each instance of entered knowledge. An internal model can be customized by locking certain slots giving special views of the unit. This is useful for reducing choices in a generalized model and improves the time for entering data.

INTRODUCTION

A knowledge-based (KB) system's interface critically affects the system's overall quality and usefulness. In practical use, the quality of the interface strongly influences (1) whether the user is initially attracted to and persists in using the system and (2) the validity with which information can be communicated between the user and the system.

During the development of the knowledge base for a KB system, similar considerations apply to the means by which knowledge is acquired for the system. In this context, the interface of a knowledge acquisition tool establishes the medium within which a domain expert carries out two critical tasks: (1) describing knowledge which constitutes the knowledge base and (2) confirming the system's representation of knowledge. The accuracy and ease with which these two tasks can be performed has a strong influence on the validity of knowledge that ultimately forms the system's knowledge base.

We believe that the KB system's validity also can be enhanced if domain experts test the system's performance incrementally as the system's knowledge base is developed. In other words, after a KB system's reasoning engine has been implemented, a domain expert can use a knowledge acquisition tool to introduce information into the knowledge base and verify its representation. Then, the domain

* Note: This work was supported by grant #411-85-0009 from the Social Sciences and Humanities Research Council of Canada and grants from Sun Microsystems, Inc. and Simon Fraser University.

expert can test the system's performance with the added information. By iterating over cycles of knowledge acquisition and testing, the domain expert can tune the knowledge base for accuracy and performance within the environment of the system's reasoning engine. This process also reflects a widely accepted principle that domain knowledge should be separated from methods for reasoning with that knowledge.

Concerns about a KB system's validity often extend beyond the development phase. Many domains in which KB systems have application are domains that accumulate knowledge continuously. Medicine, law, and education are examples. In such areas, domain knowledge grows and changes; there is a corresponding need to update and reconfigure a system's knowledge base. Without this facility, a KB system designed to work in a dynamic knowledge domain is not likely to be a long-term success

This need to update knowledge poses a problem. In the development phase of a KB system, computer specialists and domain experts are active co-participants in knowledge engineering. However, once the system is in place, the task of updating its knowledge base should be one that a domain expert can perform solo. This requires an interface that makes knowledge acquisition simple and convenient. Faneuf and Kirk (1988) have described the OASIS User Interface Management System as one approach to solving this problem.

In domains that are dynamic, where new knowledge is uncovered or current perceptions are changed, domain knowledge may not be the only information that needs updating. It also may arise that features of the representation of knowledge need to be altered and that a new interface for acquiring that knowledge must be described for the system. Since interfaces are usually related to low levels of software and hardware configurations, developing an interface for knowledge acquisition often takes a long time and demands detailed development of code by a computer specialist. This nullifies the objective of having a domain expert be able to update the knowledge base solo. Can burdens usually borne by the computer specialist be shifted partly to the system? Can the roles played by knowledge engineers and domain experts during the development phase be merged so that the domain expert could do alone what previously required the assistance of a computer expert?

This paper briefly describes the PRED interface tool. PRED allows domain experts to create appropriate frame-based knowledge acquisition interfaces for KB systems quickly and without knowledge of a programming language. These interfaces provide visibility for complex domain knowledge, speed up entering knowledge into a system's knowledge base, and allow confirmation of knowledge for accuracy. Tuning a system's knowledge base is not addressed in this paper.

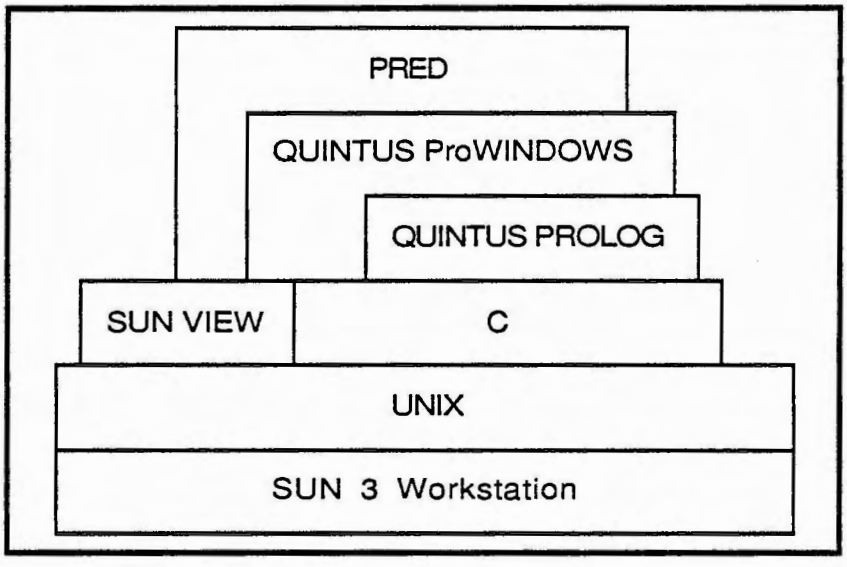
PRED provides frame editing facilities as an interface between itself and KATI (Dumaresq, Leong, Kramer, Xie and Winne, 1988), (see below) or between PRED and ENCODER (Xie, 1988). Through menu selection, the user of KATI or ENCODER can enter, search, show, display, copy and modify knowledge represented in a panel. ENCODER is our "first cut" tool to help staff translators stock a library-knowledge base with research articles about teaching. This library-knowledge base forms a cornerstone in the development of Project DOCENT (Winne and Kramer, 1988).

PRED has been used to implement primitives as low-level elements representing concepts found in research articles about teaching. These are used both in definitions by our lexicon editor and in actual article translation with ENCODER. The lexicon editor uses PRED-based definitions to create high-level terms which form an expression language for instruction (ELI). ELI provides the basis for communication with our Knowledge Acquisition Tool for Instruction (KATI) which is currently under development and has supplanted ENCODER in its objective to assist in the time-critical task of article translation.

PRED is built on the top of ProWindows 1.0 and is written in Quintus-Prolog. Quintus-Prolog is a fast and efficient implementation of the logic programming language Prolog. ProWindows is an environment available to Quintus-Prolog programmers who wish to create window-based user interfaces using Prolog.

Both Quintus-Prolog and ProWindows are developed by Quintus Computer Systems, Inc. ProWindows is based on SunView which is a multi-window user interface toolkit built on top of SunGuide - Sun General User Interface Design Environment and runs on SUN Workstations. ProWindows links an object-oriented approach with Prolog's symbolic processing abilities.

ProWindows and Prolog communicate using message passing and unification. The levels of software between PRED and computer hardware are illustrated below.



PRED'S ORGANIZATION AND MAIN COMPONENTS

Knowledge can be represented in different forms. In most commercial KB system tools, knowledge is divided into two types: factual knowledge (or data) and processes for manipulating data to solve problems (Harmon & King, 1985). In system's such as ES/P ADVISOR, KES and OPS5, facts are represented as attribute-value pairs or object-attribute-value triplets. A few large hybrid tools such as KEE, ART and SRL represent facts as frames. In all but a few commercial systems, problem-solving processes are represented by IF-THEN rules

Nau and Gray (1988), however, argue that rules are not necessarily a natural way to represent problem-solving processes, and that rule-based systems working on large knowledge bases may require excessive computational resources to match rules to the current state of a problem to be solved. Frames and their isomorphisms are deemed an especially suitable representation for complex knowledge:

> The advantages of frame languages are considerable: They capture the way experts typically think about much of their knowledge, provide a concise structural representation of useful relations and support a concise definition-by-specialization technique that is easy for most domain experts to use (Fikes & Kehler, 1985).

Thus, PRED represents all knowledge by frames. Declarative knowledge is represented by slots and slot values. Procedural knowledge is represented by demons (a concise single-purpose program attached to a particular slot) that supply or change values in other slots to accord with newly entered information. To provide a consistent interface, demons also are formatted as frames with slots for conditions and actions.

PRED supports multiple windows and menu driven panels. A panel is a dialog box containing single selection items, multiple selection items, text items, messages and buttons. In defining a panel, a unique "title" is assigned and stored as an identifier. The title can be used later to organize frames into menu-accessible groups. PRED contains the following six components (see Figure 1):

1. The interactive editor window is the interface between a domain expert and PRED. This editor allows the domain expert to define slots and their relationships thereby creating an internal model of the knowledge.
2. Syntax checking predicates analyse the internal model of knowledge and report inconsistencies in slot specifications or relationships.
3. Development predicates create the program for a panel by writing the actual ProWindows code.
4. Management predicates initiate panels and control the display of a panel's items.
5. Translating predicates create the internal dataform and the textform printout for knowledge represented in a completed panel.
6. Editing predicates search, show, display, copy and modify knowledge represented in a panel.

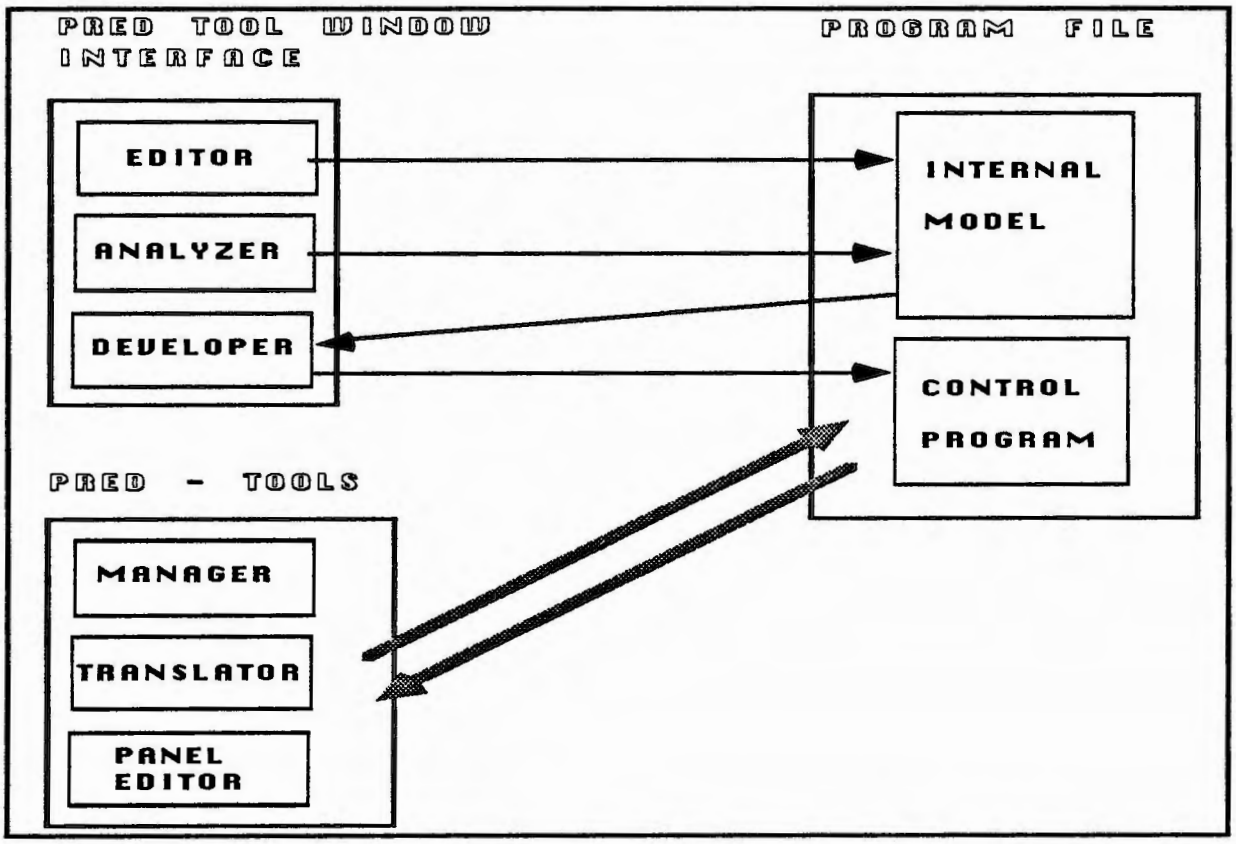


Figure 1: Organization of the Six Components of PRED

PRED creates internal models that correspond to frames and stores them in a knowledge base as Prolog facts. Models are defined as binary relations named 'dictionary object' between an identifier and a definition of the panel that a domain user will use to instantiate knowledge. Optionally, a popup window and demons can be included in an internal model. These models, or schemata for knowledge, contain specifications of slot labels, slot types, default values and the relationships between slots and sub-slots.

Internal models also are used in the control of the display of slots, by supplying information about the slot to the management predicates, and in the creation and explanation of the internal dataform corresponding to a frame. They facilitate both display control and dataform creation (see below) and explanation by providing a reference point from which an instance may be derived or expanded. Internal models ultimately support the inference process by allowing access to all slots in the completed frame in much the same way a map helps a traveller find her way in a new city.

To create a "panel presentation" of a frame of knowledge, PRED requires that domain experts translate their knowledge into primitives (elementary units) and distinguishing characteristics. A corresponding panel then can be created using PRED. Knowledge entered through panels and converted into a dataform is organized topically in a knowledge base.

To lessen storage space and facilitate the work of inferencing mechanisms, panels in which knowledge has been instantiated are stored in a compact and concise format termed a dataform (see Appendix). A dataform is a Prolog fact which defines a relation, named by the panel identifier, between the panel title, a single-digit code (for expansion), and a list of active slot-values with their internal model positions numerically specified.

The dataform is created by extracting the values from a panel and placing them in a two-member list formatted to contain information about which slots are active (which can be displayed) and which contain information. This is achieved by giving the list two members: the first is a decimal equivalent of the binary evaluation of the second member which also is a list.

The binary evaluation represents active slots and slot daughters (slot areas). Slots which have been locked out (inactive) are considered binary zeroes and do not take up space in the list with any representation. Slot areas are considered as binary ones and consist of either a string or another list using the same two-member format. The ones and zeroes are converted from binary to a decimal figure and become the first member of the two-member list. A sample frame is given as an example:

```
Label: sample                    [7,
slot1: a              1          [a,
slot2: b              1           b,
multiple-slots        1              [2,
  m1:                   0            [
  m2: c                 1             c
  m3:                   0            ] ]]]
```

The resulting two-member list format is [7, [a, b, [2, [c]]]]. If multiple-slots was inactive it would be counted as a zero giving a binary result of 110 which is decimal six. The format for this configuration would be [6, [a, b, [0 [ ]]]].

We consider this method of mapping provisional because of the design flaw inherent in position representation. This method does not allow changes to be made to a panel without having to replace all related dataform instantiations with the modified positional equivalents.

Once an internal model is created it can be customized to capture a particular aspect of the knowledge. This is achieved by locking slots in one of three ways: (1) the user sees the slot and has the option to provide a value, (2) the user sees the slot and must provide a value, or (3) the user does not see the slot. This simplifies displays for otherwise complex views of information and facilitates the entry of data by hiding irrelevant slots.

PRED's frame editing facilities are activated through menu selection allowing the user to modify entries in the knowledge base. These facilities are quite different from those of the interactive editor window because they are represented by editing-mode buttons at the foot of each panel and give the user the ability to add, change or redo the

contents of a frame. For example, the end-user may indicate that a frame has been filled by selecting the 'Done' button. This causes PRED's management and translating predicates to record the dataform in its internal knowledge base according to one of four categories which are determined from the frame identifier. These categories are menu, frame, structure or demon (see below). A text version of the knowledge is then generated and can be displayed in a working window or printed as hard-copy.

In sum, these facilities equip PRED as an interface development tool which domain experts can use on their own for creating specialized frame-based interfaces by which they can easily acquire, edit, and update knowledge for a KB system.

THE USER'S INTERFACE FOR PRED

Our intent is that PRED not require the user to have a computer background. At the present, however, the interface contains much "framers" lingo quite foreign to non-computer specialists. This will be reduced as much as possible for the next version.

The PRED Tool Window consists of 3 subwindows: Edit Window, Display Window and Graphic Window (see Figure 2).

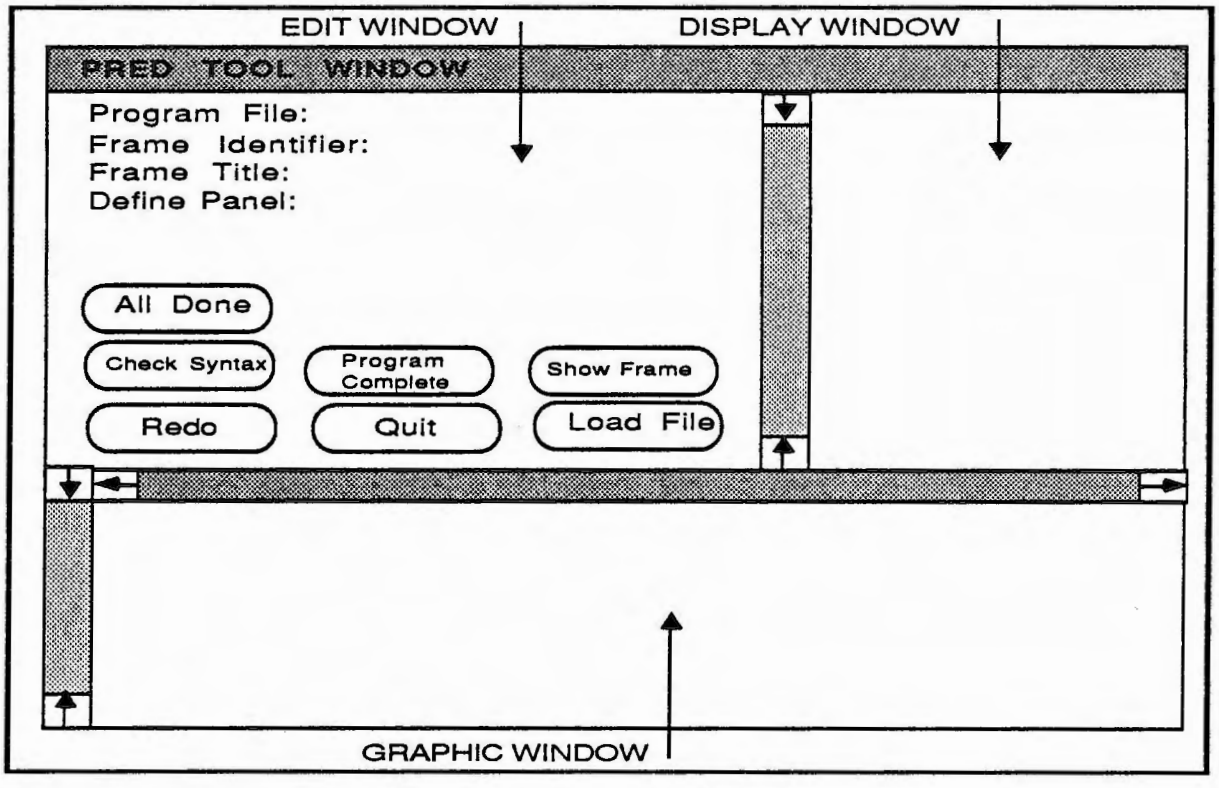


Figure 2: The Three Windows of PRED

Edit Window is a dialog box containing several control buttons. In this window, users edit a frame and specify:

1. the 'Program File' where PRED stores:
   - the created internal model of the knowledge,
   - the panel creation program
   - and the panel item (i.e. slot) control program
2. the 'Frame Identifier' which is the key for creating, storing and searching the frame
3. the 'Frame Title' which is used as a secondary key for organizing groups of panels within menus
4. the currently edited portion of the internal model using 'Define Panel'
5. the type, label and default values of a slot, and the relationship between slots.

In version 1.0 of PRED, a frame may be defined to have a pop-up menu and/or invoke the Data Entry Form demon by selecting the 'Define Popup' and 'Define Called Data Entry Forms' steps respectively.

Display Window, as its name suggests, displays the results of each action selected by the user. For example, when the 'All Done' button is pressed in Edit Window, the edited internal model is shown in the Display Window and a copy is sent to the specified Program File. Selecting the 'Check Syntax' button invokes the analyser which checks for exceptions in the syntax of the internal model and prints a summary of exceptions found. An annotated 'pretty print' of the model contents is also sent to the Display Window. Furthermore, the analyser invokes a graphic display program which draws a tree diagram in Graphic Window that depicts the relationships between the frame's slots. By correcting exceptions and confirming the relationships, users may ensure that the structure of the frame is correct.

When the 'Program Complete' button is pressed, the necessary programs for controlling panel items are created and appended to the specified Program File. This file is shown in the Display Window. Finally, when the 'Show Frame' button is pressed, the system creates a panel from the Program File and displays it to the user who may then make further modifications, if necessary.

CREATING FRAMES

Each frame must have an identifier which may start with the letter 'm', 'f', 's' or 'd' indicating menu, frame, structure (i.e., frame with a popup) or data entry demon, respectively. The remaining letters of the identifier are optional and may contain the following characters: a-z, 0-9 and '_'.

To define a frame the user normally follows these steps: define Slots, define Popups, and define Data Entry Form demons. Minimally, the user may select define Slots. Each step is indicated by selecting the 'Define Panel' button in the Edit Window.

To define a slot the 'Slot Type' and the 'Relation with the previous slot' must be specified. Each type is translated into a ProWindows menu object; slot types and their definitions follow. ProWindows terminology is shown in square brackets (further information can be found in the Quintus ProWindows documentation):

1. ' ' (space) [N/A]
   This indicates a null slot
2. linear layout [linear_layout]
   The following group of slots will
   appear on the panel in a linear fashion descending vertically
3. single selection [cycle]
   Place the following group of values into a menu
   which displays only the top item as a default (may be null).
   The user may click through each item individually exposing
   the value and perhaps activating associated processes
   (slots or demons). Or, using an alternative mousing method,
   display the entire list of values and 'slide' down to select a value
4. multiple selection [toggle]
   Place the following groups of values into a menu
   which displays every item. The user may select/deselect
   any number of items in the list thereby assigning multiple
   slot values and/or activating associated processes
5. text [text_item]
   Display a default value and allow the user to change it
   if necessary. The default value may be blank and the
   text_item may have an associated process attached
6. message [label]
   Print a text value which may not be altered by the user
   and has no associated processes
7. button [button]
   Display a button with a given label which initiates an
   associated process when selected.

In general, associated processes are activated by the user who selects a value provided as a default, activating a demon or uncovering another set of slots.

A slot's relationship with a previous slot specifies a family relationship in the frame's tree structure. The relationship may be one of:

- ' ' (space), this is used for indicating a null slot
- 'daughter',
- 'sibling',
- 'aunt',
- 'grand-aunt'
- 'great grand-aunt'.

Daughter and sibling are used to indicate a descent into the tree where slot groups are being nested. Aunt refers to the relationship between the last sibling in a branch and a sibling to the original daughter and are used to indicate a return from a nested group.

The 'Slot Label' can be any string as defined above, and is displayed as an attribute of the frame.

The 'Slot Contents' for single selection and multiple selection allow the user a choice of pre-set values. The values must be placed in a list bounded by square parentheses and may contain any number of strings. In the case of single selection, the first item in the list becomes a default value. A null (empty space) as the first item means there is no default.

If an associated process is to be attached to a single or multiple selection item then a corresponding daughter must be defined and each item in the contents list must be matched with a sibling slot definition. In many cases only certain item selections will be required to activate other processes and these must have slot definitions as corresponding siblings. Those that do nothing except supply their value to the slot must have a space as a corresponding sibling. The space is translated by PRED into an underscore as the model is constructed. After each slot is defined, PRED also generates a unique slot identifier and places it before the label of the definition.

A frame may have a browsing window, a scrollable menu-like window, as its popup. This allows the creator of the frame to relate certain slots to text files and give the end-user access to these files for importing values during a knowledge entry session. The user specifies a popup for a frame by selecting 'Define Panel' as 'Define Popup', and specifying the 'Popup Content' and the 'Related Item'. Each 'Popup Content' entry is a quoted string consisting of a slot label separated by a colon and the title of a frame. The 'Related Item' entry must be the corresponding slot identifier preceding the slot label and shown in Display Window. For example,

```
[popup,
    ['Presentation: MATERIALS',
    'Person responding: DEMOGRAPHICS',
    'Response ascribed to: DEMOGRAPHICS',
    'Content presented for evaluation: NON ACHIEVEMENT QUESTION
    CONTENT',
    'Format and scoring: NON ACHIEVEMENT TEST SCORING' ],
[@sntd_5,@sntd_6,@sntd_7,@sntd_8,@sntd_9] ]
```

can be defined as follows:

1. Click the right button of the mouse on 'Define Panel' in the Edit Window to select the option 'Define Popup'
2. Type 'Presentation: MATERIALS' in item 'Popup Content: ' and put @sntd_5 in item 'Related Item: '
3. Click button 'One Popup Done'
4. Repeat 2 and 3 for the remaining popup items and then click button 'All Popups Done'.

A Menu can be viewed as a special kind of frame, that only has a 'menu' type slot. PRED will control the display of the menu and translate menu selections into Prolog calls but the called programs must be inserted in the Program File by the user.

Creating Demons

The frames created by PRED may have two kinds of demons: Data Entry Forms and User Defined Demons. The Data Entry Forms can be invoked by either a 'single selection' or 'multiple selection' type slot and return a value to a message type slot (see CREATING FRAMES) in the invoking frame. The User Defined Demons may be invoked by a 'single selection', 'multiple selection', 'text', or 'button' slot, but it is the user's responsibility to insert a corresponding program into the program file to pick up the Prolog call.

In PRED we have provided the built-in Data Entry Forms listed in Table 1. The detail of the format and semantic meanings of these built-in Data Entry Forms are listed in the BLEEP Manual (Kramer, Winne, Roberts, Leong, Kanevsky, Seifert and Reid, 1988). An example is shown in Figure 3.

Name of Data Entry Forms	Identifier
'ADJECTIVE'	dads
'DIRECTION'	ddis
'DURATION'	ddus
'EQUATION BUILDER'	debs
'MODIFIERS'	dmos
'NUMBER'	dnus
'PHYSICAL DIMENSION'	dphs
'SCALES'	dscs
'STATISTICS'	dsts
'TIMING'	dtis

Table 1:The built-in Data Entry Forms

```
dictionary_object(dmos,
[[_, linear_layout, _, _],
  [[@dmos_mod, cycle, 'More/less? ',[' ','~', '<', '>']],
      [@dmos_cho, cycle, 'Modifiers: ', [' ', 'Appropriateness',
          'Attentiveness',
          'Competence/Ability', 'Correctness', 'Difficulty',
          'Duration', 'Effort', 'Intensity/degree', 'Likelihood',
          'Loudness', 'Pace', 'Quality', 'Relative Frequency',
          'Spontaneity/creativity', 'Timing',
          'Usefulness', 'Value/importance']],
      [@dmos_deg, cycle, 'Degree: ', [' ', 1, 2, 3, 4, 5]]]]).
```

Figure 3: Internal Model of Data Entry Form: MODIFIERS/dmos

Users may refer to these Data Entry Forms to create their own demon in order to provide a specialized response to a particular slot selection.

Modification of the Panel Layout

To modify the layout of a panel, the user can bring the program file into Display Window and modify the item positions by putting 'point(X, Y)' into the items, where X and Y are the coordinates of an item within the panel and X is the product of 22 and the approximate line number. After the modification, the program can be saved by using the popup menu (provided by SunView) in Display Window by pressing the right key on the mouse. The modified panel can then be displayed by clicking 'Show Frame' button in Edit Window. An improved layout modification mechanism will be developed in a later version of PRED.

Example Of Frame Creation

The following example describes the steps of creating a frame and its display panel. The example is selected from the ongoing research project DOCENT which is an artificially intelligent planning system for novice teachers. The example given will deal with the ATTRIBUTION frame (see Figure 4). Its identifier is 'fats' and two of its slots will call the Data Entry Form demon 'MODIFIERS'

In order to create a frame, an internal model must first be created. To do this, the user first specifies the 'Frame Identifier', 'Frame Title' and 'Program File'. These are:

- 'Frame Identifier': fats
  signifies 'f' for frame, 'at' for 'attribution' and 's' for the creator's surname:'shunen'
- 'Frame Title': ATTRIBUTION
- 'Program File'.

The user proceeds by creating the second argument of the internal model that contains the actual domain knowledge and is completed by selecting the 'Define Panel' as 'Define Slots'.

```
dictionary_object(fats,
[[_, linear_layout, _, _],
    [[@fats_1, text_item, 'Indescribable: ', _],
    [@fats_2, cycle, 'Unspecified in the study: ',[' ',yes]],
    [@fats_3, cycle, 'Global: ', [' ',yes]],
    [[@fats_4, cycle, 'Outcome: ',  [' ', 'success/pass/mastery',
            failure, 'DE_MODIFIER: ']],
            [_, _, _, [@fats_5, label, '', _]]],
    [@fats_6, toggle, 'General Dimensions - outcome due to: ',
        ['internal to actor', 'in situation/external to actor',
        'controllable by actor', 'not controllable by actor',
        'stable over time', 'unstable/erratic over time']],
    [[@fats_7, toggle, 'Specific Dimensions - outcome due to: ',
        ['ability', 'effort', 'skill','others','luck',
        'fatigue','mood','task difficulty',
        'involvement by other person: ']],
```

Figure 4: The ATTRIBUTION Internal Model

```
        [_, _, _, _, _, _, _, _,
        [[_, linear_layout, _, _],
            [[@fats_8, cycle, 'person: ', [' ','peer','teacher']],
    [@fats_9, cycle, 'type of involvement: ',
                [' ', 'general supportive talk',
                'direct help with task']]]]],
    [[@fats_10, cycle, 'Typical: ', [' ', yes, no, 'DE_MODIFIER: ']],
        [_, _, _, [@fats_11, label, '', _]]],
    [@fats_12, cycle, 'Situational Generalizability: ', [' ',
            'peculiar to this situation',
            'generally common to most situations']]]],
[popup, [], []],
[[@fats_5, @fats_11], [dmos, dmos]]).
```

Figure 4: The ATTRIBUTION Internal Model

To construct the slots of the internal model as shown above, the user begins thus:

1. Select the 'Slot Type'
2. Select the 'Relationship with the previous slot'
3. Give the 'Slot Label', optional
4. Specify the 'Slot Contents', optional
5. Click the button 'One Slot Done'.

The above five steps are repeated for each slot until all slots are specified. At this point the 'All Slots Done' button is selected. This particular example did not require a popup and therefore is given the argument of 'popup' and two empty lists: [popup, [], []]. This argument may also be simply an underscore meaning 'no value expected'.

To create the final argument [[@fats_5, @fats_11], [dmos, dmos]] which corresponds to the invocation of the MODIFIERS Data Entry Forms, select the 'Define Panel' as 'Define Called Data Entry Forms', then:

1. specify that the 'DEF Result Item' will be entered from the slot @fats_5 (the slot identifier which is shown in the Display Window) and that the 'DEF' to be used is dmos (the demon identifier) then click the button 'One DEF Done'
2. specify the 'DEF Result Item' as @fats_11 and the 'DEF' as dmos, click the button 'One DEF Done'
3. Click buttons 'All DEF Done' and 'All Done'.

The final version of the internal model will be shown in the Display Window.

After constructing the internal model the user should analyze the model layout with the syntax checker. Selecting the 'Check Syntax' button, will give both a text based and a graphic analysis in the Display Window and Graphic Window respectively. The

graphic analysis is given in the form of a tree representation of the frame. A portion of the text based display is shown in Figure 5 below.

```
Analyzing model syntax:
null, linear_layout: null
    ->@fats_1, text_item: Indescribable:
    ->@fats_2, cycle: Unspecified in the study:
    ->@fats_3, cycle: Global:
    @fats_4, cycle: Outcome:[
,success/pass/mastery,failure,DE_MODIFIER: ]
    [ ]
    null here
    [success/pass/mastery]
    null here
    [failure]
    null here
    [DE_MODIFIER: ]
        ->@fats_5, label:
    ->@fats_6, toggle: General Dimensions - outcome due to:
    .
    .
    .
    @fats_7,toggle:Specific Dimensions - outcome due to:
                [ability,effort,skill,others,luck,fatigue,mood,task difficulty,
                 involvement by other person: ]
    [ability]
    null here
    [effort]
    null here
    [skill]
    null here
    [others]
    null here
    [luck]
    null here
    [fatigue]
    null here
    [mood]
    null here
    [task difficulty]
    null here
    [involvement by other person: ]
    null, linear_layout: _4720
        ->@fats_8, cycle: person:
        ->@fats_9, cycle: type of involvement:
```

Figure 5: Textual graph of Internal Model Analyses (continued next page)

```
    @fats_10, cycle: Typical: [ ,yes,no,DE_MODIFIER: ]
    [ ]
    null here
    [yes]
    null here
    [no]
    null here
    [DE_MODIFIER: ]
        ->@fats_11, label:
    fats_12, cycle: Situational Generalizability:

Analyse popup syntax:
No Predefined Label List

Analyse dataform syntax:
    @fats_5  -> dmos (FOUND)
    @fats_11  -> dmos (FOUND)

Summary of Exceptions:
0 exceptions in total
```

Figure 5: Textual graph of Internal Model Analyses

Once the model has been created, the user may select the button 'Show Frame', in order to display the actual panel.

COMPARISON OF METHODS

Creating a knowledge acquisition interface usually entails extensive programming time. During the development of our project, we tried to create frame interfaces such as the example presented in Section IV, by SunView, ProWindows and PRED, respectively. The amount of time it took for a skilled programmer to develop the ATTRIBUTION panel using each method are listed below:

By SunView	4-7 days
By ProWindows	2-3 days
By PRED	approximately 10 minutes

By using PRED, not only is the time drastically reduced, but the debugging process is also made very simple. A domain expert can create the frame interfaces by herself as fast as a skilled programmer. By using PRED, more than 70 frame-like interfaces have been created for KATI.

CONCLUSIONS

PRED is an interface development tool which allows domain experts to easily create frame-based knowledge acquisition interfaces for an expert system in their domain. These interfaces provide a convenient means of viewing complex domain knowledge.

PRED has been developed and used to create more than 70 panel interfaces for KATI - a Knowledge Acquisition Tool for Instruction that is currently under development.

To create a panel representing knowledge as a frame, PRED requires that domain experts describe the attributes and associated values of the knowledge in its tool window.

Once panels are created they can be processed consistently by the system. A panel which has knowledge entered into it is stored in a compact and concise way to lessen storage space and facilitate the work of inference mechanisms.

Our most immediate concern is the method of organizing slot information within the dataform. The current method is to build a list of active slot-values with their internal model positions numerically specified. This does not allow changes to the panel to be made easily.

A domain expert may use PRED to develop a frame-like interface in a few minutes. This kind of work usually requires a computer programmer to program and debug for hours or even days depending on the complexity of the frame and whether ProWindows or SunView is used. Since PRED has been implemented in Prolog, a major portion could be easily integrated with other expert system tools written in Prolog.

REFERENCES

Dumaresq, David F., Leong, Jenny, Kramer, Laurane L., Xie, Shun-en and Winne, Philip H. (1988). "KATI: A Knowledge Acquisition Tool for Instruction", personal communication, Instructional Psychology Research Group, Simon Fraser University., Burnaby, B.C., Canada.

Fikes, Richard and Kehler, Tom, (1985). "The Role of Frame-based Representation in Reasoning" in Special Section, Communications of the ACM, Sept. 1985, Vol 28 No. 9, Copyright 1985

Franeuf, R. and Kirk, S., (1988). "A UIMS for building metaphoric user interfaces", in Expert Systems: The User Interface, J.A. Hendler, Ed., Ablex Pub. Corp., pp. 183-218.

Harmon, P. and King, D., (1985). "Expert Systems", John Wiley & Sons, Inc.

Kramer, L. L., Winne, P. H., Roberts, S., Leong, J., Kanevsky, L., Seifert, T., Reid, B. (in preparation, 1988). "The Basal Language for Encoding Educational Processes: BLEEP (Project DOCENT Research Report No. 2). IPRG, Simon Fraser University.

Nau, D. and Gray, M., (1988). "Hierarchical knowledge clustering: A way to represent and use problem-solving knowledge", in Expert Systems: The User Interface, J.A. Hendler, Ed., Ablex Pub. Corp., pp. 81-98.

Winne, P. and Kramer, L.L., (1988). "Representing and inferencing with knowledge about teaching: DOCENT - An artificially intelligent planning system for teachers", in Proceedings of the international Conference on Intelligent Tutoring Systems, Montreal, Quebec.

Xie, S., (1988). "ENCODER User's Guide", IPRG, Simon Fraser University,Burnaby, B.C., Canada.

COMPANIES

ES/P ADVISOR is a trademark of Expert Systems, Ltd., King of Prussia, PA.
KES is a trademark of Software Architecture & Engineering, Arlington, VA.
OPS5 is a trademark of Digital Equipment Corp., Concord, MA.
KEE is a trademark of Intellicorp, Menlo Park, CA.
ART is a trademark of Inference Corp., Los Angeles, CA.
SRL is a trademark of Carnegie Group, Pittsburgh, PA.
ProWindows is a trademark of Quintus Computer Systems, Inc.
Quintus Prolog is a trademark of Quintus Computer Systems, Inc.
Sun Workstation is a registered trademark of Sun Microsystems, Inc.
SunView is a registered trademark of Sun Microsystems, Inc.

APPENDIX

Internal Data Form of an instantiated ATTRIBUTION Frame

```
fats(attri_1,1,[252,[1,[0,_176],17,[0,[]],[3,'[appropriateness, 3]'],2]]).
```