Proceedings of the 4TH KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre
Banff, Alberta, Canada
October 1-6, 1989

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
4TH KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff Conference Centre, Banff, Alberta, Canada
October 1-6, 1989

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.

4TH KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

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

CONTENTS

Program iv

List of Participants and Authors ix

Technical Papers (listed alphabetically by first author)

Knowledge for Design 1-1
Thomas R. Addis, University of Reading, U.K.

A Mediating Representation to Assist Knowledge Acquisition with MACAO 2-1
Nathalie Aussenac, Jean Frontin, Marie-Helene Riviere, Jean-Luc Soubie, Universite Paul Sabatier, France

Acquiring and Verifying Control Knowledge for a Blackboard System 3-1
Lawrence S. Baum, David B. Shema, John H. Boose, Jeffrey M. Bradshaw, Boeing Advanced Technology Center, USA

Learning Flexible Concepts through a Search for Simpler but Still Accurate Descriptions 4-1
F. Bergadano, Universita di Torino, Italy; Stan Matwin, University of Ottawa, Canada; R. S. Michalski, J. Zhang, George Mason University, USA

From ETS to AQUINAS: Six Years of Knowledge Acquisition Tool Development 5-1
John H. Boose, Jeffrey M. Bradshaw, Catherine M. Kitto, David B. Shema, Boeing Advanced Technology Center, USA

The Block Representation in Knowledge Acquisition for Computer Integrated Documentation 6-1
Guy A. Boy, NASA-Ames Research Center, USA

KWACQ'ing with DDUCKS: Knowledge Acquisition Techniques for Decision Analysis Using Axotl and Aquinas 7-1
Jeffrey M. Bradshaw, Stanley P. Covington, Peter J. Russo, John H. Boose, Being Advanced Technology Center, USA

Structured Matching: A Task-Specific Technique For Making Decisions 8-1
Tom Bylander, Todd R. Johnson, Ohio State University; Ashok Goel, Georgia Institute of Technology, USA

A Framework For The Comparative Evaluation Of Knowledge Acquisition Tools and Techniques 9-1
Jasbir Singh Dhaliwal, Izak Benbasat, University of British Columbia, Canada

Hypothesis Testing: An Appropriate Task for Knowledge-Based Systems 10-1
B.Z. Feldman, Paul Compton, G. A. Smythe, Garvan Institute of Medical Research

An Approach to Knowledge Acquisition Based on the Structure of Personal Construct Systems 11-1
Kenneth M. Ford, University of West Florida; Jack R. Adams-Webber, Brock University; Frederick E. Petry, Tulane University; Paul J. Chang, Stanford University, USA

Design Requirements for Knowledge Support Systems 12-1
Brian R. Gaines, University of Calgary

A Heterogeneous Knowledge Support System 13-1
Brian R. Gaines, University of Calgary; Alain Rappaport, Neuron Data; Mildred L. G. Shaw, University of Calgary

CLASSIKA: A Knowledge Acquisition Tool for Use by Experts 14-1
Ute Gappa, Institute of Logic, West Germany

Knowledge Acquisition From Natural Language For Expert Systems Based On Classification Problem-Solving Methods 15-1
Fernando Gomez, University of Central Florida, USA

Knowledge Dependencies 16-1
Chris Jacobson, Mike Freiling, Tektronix, Inc., USA

Bringing Corporate Knowledge Into Focus with CAMEO 17-1
William P. Jones, Arthur D. Little, Inc., USA

Graphical Knowledge Acquisition 18-1
Robert L. Joseph, Carnegie-Mellon University, USA

Analogy in Knowledge Acquisition: Description of an Analogical Tool 19-1
Debbie Leishman, University of Calgary

Empirical Knowledge Refinement in Noisy Domains 20-1
Ting-peng Liang, University of Illinois at Urbana-Champaign, USA

A Methodology for Knowledge Acquisition In A Group Decision Support System Environment 21-1
Yihwa Irene Liou, E. Sue Weber, Jay F. Jnamaker, Jr., University of Baltimore, USA

An Integrated Toolkit for Cognitively-Based Knowledge Acquisition 22-1
Karen L. McGraw, Juliana S. Lancaster, The BDM Corporation, USA

A Practical Program For Handling Bias In Knowledge Acquisition 23-1
Mary A Meyer, Jane M. Booker, Los Alamos National Laboratory, USA

Hypermedia in Knowledge Acquisition and Specification of User Interface for KBS: An Approach and a Case Study 24-1
Jukka Rantanen, Technical Research Centre of Finland (VTT), Finland

Sortal Analysis with SORTAL, a Software Assistant for Knowledge Acquisition 25-1
Stephen Regoczei, Trent University, Canada ; Graeme Hirst, University of Toronto, Canada

PM: A Metrics Driven Plan Compiler 26-1
Robert G. Reynolds, Jonathan I. Maletic, Stephen E. Porvin, Wayne State University, USA

Knowledge Acquisition as Cognitive Modeling 27-1
Valerie L. Shalin, John L. Bloomfield, Honeywell Systems and Research Center, USA, Peter J. Bullemer, Honeywell Corporate Systems Development Division, USA

Knowledge Acquisition for Advice Systems: FUSION, A Network Cognitive Support System 28-1
Maurice Sharp, University of Calgary, Canada

Mental Models in the Knowledge Acquisition Process 29-1
Mildred L. G. Shaw, J. Brian Woodward, University of Calgary

Design Knowledge Capture and Alternative Generation Using Possibility Tables in CANARD 30-1
David B. Shema, Jeffrey M. Bradshaw, Stanley P. Covington, and John H. Boose, Boeing Advanced Technology Center, USA

A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis 31-1
Douglas Skuce, Wang Shenkang, Yves Beauvillé, University of Ottawa, Canada

Using a Knowledge Acquisition Tool to Test Strategy Variations 32-1
Jeffrey Stout, Linda Green, and Sandra Marcus, Boeing Advanced Technology Center, USA

Understanding Metaphors By Frustration-Based Learning Method 33-1
Masaki Suwa, Hiroshi Motoda, Hitachi, Ltd, Japan

Knowledge Acquisition By Abductive And Inductive Explanation 34-1
Hirokazu Taki, ICOT, Japan

New Learning Algorithm For Rule Extraction By Neural Network And Its Application 35-1
Jun Tani, Toshikazu Hirobe, Kazuo Niida, Ichirou Koshijima, Chiyoda Corporation; Humio Murakami, Nippon Felt Co., Ltd, Japan

A Knowledge Acquisition Perspective On Knowledge-Level Models 36-1
Bob Wielinga , University of Amsterdam; Hans Akkermans, Netherlands Energy Research Foundation; Guus Schreiber, University of Amsterdam; John Balder, Netherlands Energy Research Foundation

PROGRAM
4TH AAAI-SPONSORED KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Sponsored by the American Association for Artificial Intelligence
Banff, Alberta, Canada, October 1-6, 1989

Co-Chairmen

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

Program Committee

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

DAILY SCHEDULE

Workshop sessions will be held in the residence building. Continental breakfasts will be available before each session in the workshop area at 7:30. Sessions will start each day at 9:00. There will be coffee and juice breaks each morning and afternoon, and lunch will be served from 12:00 - 1:00, also in the workshop area. Hot breakfasts will be available in the cafeteria in a separate building. There will be group dinners Monday, Tuesday, and Thursday evenings in a separate Centre building.

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

SCHEDULE OF EVENTS

Sunday, October 1

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

Monday, October 2

Overview

9:00 - 9:40 Workshop Welcome and Introduction - Boose & Gaines
9:40 - 10:20 Design Requirements for Knowledge Support Systems
Brian Gaines, University of Calgary (discussion: Alain Rappaport)

10:20 - 10:40 Break

Cognition and Structure

10:40 - 11:20 A Practical Program For Handling Bias In Knowledge Acquisition
Mary A Meyer, Jane M. Booker, Los Alamos National Laboratory, USA (discussion: Paul. Compton)

Posters
11:20 - 12:00 Informal Poster Kickoff Session

Knowledge: Representation and Roles
Knowledge for Design
Thomas R. Addis, University of Reading, U.K.
A Mediating Representation to Assist Knowledge Acquisition with MACAO
Nathalie Aussenac, Jean Frontin, Marie-Helene Riviere, Jean-Luc Soubie, Universite Paul Sabatier, France
Acquiring and Verifying Control Knowledge for a Blackboard System
Lawrence S. Baum, David B. Shema, John H. Boose, Jeffrey M. Bradshaw, Boeing Advanced Technology Center, USA
The Block Representation in Knowledge Acquisition for Computer Integrated Documentation
Guy A. Boy, NASA-Ames Research Center, USA
Knowledge Dependencies
Chris Jacobson, Mike Freiling, Tektronix, Inc., USA
Graphical Knowledge Acquisition
Robert L. Joseph, Carnegie-Mellon University, USA
Hypermedia in Knowledge Acquisition and Specification of User Interface for KBS: An Approach and a Case Study
Jukka Rantanen, Technical Research Centre of Finland (VTT), Finland
PM: A Metrics Driven Plan Compiler
Robert G. Reynolds, Jonathan I. Maletic, Stephen E. Porvin, Wayne State University, USA
Design Knowledge Capture and Alternative Generation Using Possibility Tables in CANARD
David B. Shema, Jeffrey M. Bradshaw, Stanley P. Covington, and John H. Boose, Boeing Advanced Technology Center, USA

Cognition and Structure
Knowledge Acquisition as Cognitive Modeling
Valerie L. Shalin, John L. Bloomfield, Honeywell Systems and Research Center, USA, Peter J. Bullemer, Honeywell Corporate Systems Development Division, USA
Knowledge Acquisition for Advice Systems: FUSION, A Network Cognitive Support System
Maurice Sharp, University of Calgary, Canada

Knowledge Acquisition and Machine Learning
Learning Flexible Concepts through a Search for Simpler but Still Accurate Descriptions
F. Bergadano, Universita di Torino, Italy; Stan Matwin, University of Ottawa, Canada; R. S. Michalski, J. Zhang, George Mason University, USA
Analogy In Knowledge Acquisition: Description of an Analogical Tool
Debbie Leishman, University of Calgary
Knowledge Acquisition By Abductive And Inductive Explanation
Hirokazu Taki, ICOT, Japan

Knowledge Acquisition Tools, Methods, and Techniques
From ETS to AQUINAS: Six Years of Knowledge Acquisition Tool Development
John H. Boose, Jeffrey M. Bradshaw, Catherine M. Kitto, David B. Shema, Boeing Advanced Technology Center, USA
An Approach to Knowledge Acquisition Based on the Structure of Personal Construct Systems
Kenneth M. Ford, University of West Florida; Jack R. Adams-Webber, Brock University; Frederick E. Petry, Tulane University; Paul J. Chang, Stanford University, USA
A Heterogeneous Knowledge Support System
Brian R. Gaines, University of Calgary; Alain Rappaport, Neuron Data; Mildred L. G. Shaw, University of Calgary
CLASSIKA: A Knowledge Acquisition Tool for Use by Experts
Ute Gappa, Institute of Logic, West Germany
An Integrated Toolkit for Cognitively-Based Knowledge Acquisition
Karen L. McGraw, Juliana S. Lancaster, The BDM Corporation, USA

12:00 - 1:00 Lunch

Knowledge: Representation and Roles

1:00 - 1:40 Design Knowledge Capture for a Corporate Memory Facility
John Boose, Jeffrey Bradshaw, David Shema, Stanley Covington, Boeing Advanced Technology Center (discussion: Ken Ford)

1:40 - 2:20 A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis
Douglas Skuce, Wang Shenkang, Yves Beauvillé, University of Ottawa, Canada (discussion: Bob Reynolds)

2:20 - 2:40 Break

Cognition and Structure

2:40 - 3:20 Hypothesis Testing: An Appropriate Task for Knowledge-Based Systems
B.Z. Feldman, Paul Compton, G. A. Smythe, Garvan Institute of Medical Research (discussion: Ute Gappa)

3:20 - 4:00 Mental Models in the Knowledge Acquisition Process
Mildred L. G. Shaw, J. Brian Woodward, University of Calgary (discussion: Tom Addis)

7:00 Dinner buffet, full cash bar

Tuesday, October 3

Knowledge: Representation and Roles

9:00 - 9:40 A Knowledge Acquisition Perspective On Knowledge-Level Models
Bob Wielinga , University of Amsterdam; Hans Akkermans, Netherlands Energy Research Foundation; Guus Schreiber, University of Amsterdam; John Balder, Netherlands Energy Research Foundation (discussion: Chris Jacobson)

9:40 - 10:20 Using a Knowledge Acquisition Tool to Test Strategy Variations
Jeffrey Stout, Linda Green, and Sandra Marcus, Boeing Advanced Technology Center, USA (discussion: Bill Clancey)

10:20 - 10:40 Break

Knowledge Acquisition and Decision Making

10:40 - 11:20 A Methodology for Knowledge Acquisition In A Group Decision Support System Environment
Yihwa Irene Liou, E. Sue Weber, Jay F. Jnamaker, Jr., University of Baltimore, USA (discussion: Valerie Shalin)

11:20 - 12:00 Decision Analysis and Knowledge Acquisition
Max Henrion, Rockwell Science Center (discussion: Jeff Bradshaw)

12:00 - 1:00 Lunch

1:00 - 1:40 KWACQ'ing with DDUCKS: Knowledge Acquisition Techniques for Decision Analysis Using Axotl and Aquinas
Jeffrey M. Bradshaw, Stanley P. Covington, Peter J. Russo, John H. Boose, Being Advanced Technology Center, USA (discussion: Max Henrion)

Panel

1:40 - 2:20 What is the Role of Cognition and Modeling in Knowledge Acquisition?
William Clancey, Institute for Research on Learning; Mike Freiling, Tektronix; Valerie Shalin, Honeywell

2:20 - 2:40 Break

Natural Language Acquisition

2:40 - 3:20 Knowledge Acquisition From Natural Language For Expert Systems Based On Classification Problem-Solving Methods
Fernando Gomez, University of Central Florida, USA (discussion: Stan Matwin)

Knowledge: Representation and Roles

3:20 - 4:00 Structured Matching: A Task-Specific Technique For Making Decisions
Tom Bylander, Todd R. Johnson, Ohio State University, Ashok Goel, Georgia Institute of Technology, USA (discussion: Tom Gruber)

7:00 Dinner, full cash bar

Wednesday, October 4

Knowledge Acquisition and Machine Learning

9:00 - 9:40 Empirical Knowledge Refinement in Noisy Domains
Ting-peng Liang, University of Illinois at Urbana-Champaign, USA (discussion: Hirokazu Taki)

9:40 - 10:20 Understanding Metaphors By Frustration-Based Learning Method
Masaki Suwa, Hiroshi Motoda, Hitachi, Ltd, Japan (discussion: Guy Boy)

10:20 - 10:40 Break

10:40 - 11:20 New Learning Algorithm For Rule Extraction By Neural Network And Its Application
Jun Tani, Toshikazu Hirobe, Kazuo Niida, Ichirou Koshijima, Chiyoda Corporation; Humio Murakami, Nippon Felt Co., Ltd, Japan (discussion: Maurice Sharp)

Panel

11:20 - 12:00 How Can Knowledge Acquisition Support Design?
Guy Boy, NASA/AMES; Tom Addis, Reading University; Tom Gruber, Stanford

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

Thursday, October 5

Tools, Methods, and Techniques

9:00 - 9:40 Sortal Analysis with SORTAL, a Software Assistant for Knowledge Acquisition
Stephen Regoczei, Trent University, Canada ; Graeme Hirst, University of Toronto, Canada (discussion: Debbie Leishman)

9:40 - 10:20 A Framework For The Comparative Evaluation Of Knowledge Acquisition Tools and Techniques
Jasbir Singh Dhaliwal, Izak Benbasat, University of British Columbia, Canada (discussion: Brian Woodward)

10:20 - 10:40 Break

10:40 - 11:20 Bringing Corporate Knowledge Into Focus with CAMEO
William P. Jones, Arthur D. Little, Inc., USA (discussion: Jukka Rantanen)

Panel

11:20 - 12:00 How Can Theoretical Approaches Support Practical Tools?
Jack Adams-Webber, Brock University; Hans Akkermans, University of Amsterdam; Max Henrion, Rockwell Science Center

12:00 - 1:00 Lunch

Suggested Working Groups

Afternoon	Integration of ML and KA (Reporting: M. Suwa, T. Gruber)
	Progress in KA Tools (Reporting: B. Woodward, J. Bradshaw)
	Cognition and Modeling (Reporting: W. Clancey, M. Meyer)
	Hypermedia and Graphical KA (Reporting: G. Boy, D. Leishman)
	Mediating Representations (Reporting: M. Musen, T. Addis)
2:20 - 2:40	Break
7:00	Banquet Dinner with Annual Bear Tossing and Shark Snatching Mystery Guest Speaker

Friday, October 6

Working Group Presentations

9:00 - 9:30	Working Group 1
9:30 - 10:00	Working Group 2
10:00 - 10:30	Working Group 3
10:30 - 10:45	Break
10:45 - 11:15	Working Group 4
11:15 - 11:45	Working Group 5
11:45 - 12:15	Synthesis and Summary - Brian Gaines
12:15 - 1:00	Farewell Lunch and Bear Souvenirs; Business Meeting

LIST OF PARTICIPANTS AND AUTHORS

4TH AAAI-SPONSORED KNOWLEDGE ACQUISITION FOR KNOWLEDGE-BASED SYSTEMS WORKSHOP

Banff, Canada, October 1-6, 1989

Jack Adams-Webber
Brock University
Dept. of Psychology
St. Catharines, Ontario
L2S 3A1, Canada

Thomas R. Addis
Department of Computer Science
University of Reading
PO Box 220
Whiteknights
Reading, RG6 2AX U.K.

Hans Akkermans
Netherlands Energy Research Foundation ECN
P.O. Box 1
NL-1755 ZG Petten, The Netherlands

Nathalie Aussenac
Institut de Recherche en Informatique de Toulouse
Universite Paul Sabatier
118, route do Narbonne
31062 Toulouse Cedex, France

John Balder
Netherlands Energy Research Foundation ECN
P.O. Box 1
NL-1755 ZG Petten, The Netherlands

Monique Barbanson
Institute for Research on Learning
2550 Hanover Street
Palo Alto, CA, 94304, USA

Lawrence S. Baum
Information Systems Laboratory
Advanced Technology Center, 7L-64
Boeing Computer Services
Box 24346
Seattle, WA 98124 USA

Yves Beauvillé
Department of Computer Science
University of Ottawa
34 George Glinski
Ottawa, Ontario, Canada K1N 6N5

Izak Benbasat
Management Information Systems Division
Faculty of Commerce and Business Administration
University of British Columbia
2053 Main Mall
Vancouver, British Columbia V6T 1Y8 Canada

F. Bergadano
Dipartimento di Informatica
Universita di Torino
Corso Svizzera 185
Torino, Italy

John L. Bloomfield
Honeywell Systems and Research Center
3660 Technology Dr. MN65-2500
Minneapolis, MN 55418 USA

Jane M. Booker
MS F600
Statistics Group,
Los Alamos National Laboratory
Los Alamos, NM 87545 USA

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

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

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

Peter J. Bullemer
Honeywell Sensor and System Development Center
1000 Boone Ave. N.
Golden Valley, MN 55427

Tom Bylander
Laboratory for Artifical Inteligence Research
Department of Computer and Information Science
2036 Neil Avenue Mall
The Ohio State University
Columbus, OH 43210-1277

Paul J. Chang
Stanford University

William Clancey
Institute for Research on Learning
2550 Hanover Street
Palo Alto, CA, 94304, USA

Paul Compton
Garvan Institute of Medical Research
St Vincent's Hospital, Victoria St.
Darlinghurst
Sidney, NSW 2010
Australia

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

Jasbir Singh Dhaliwal
Management Information Systems Division
Faculty of Commerce and Business Administration
University of British Columbia
2053 Main Mall
Vancouver, British Columbia V6T 1Y8 Canada

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

Mike Freiling
Computer Research Laboratory
Tektronix Laboratories
Tektronix, Inc.
Box 500 MS 50-662
Beaverton, OR 97077 USA

Jean Frontin
Institut de Recherche en Informatique de Toulouse
Universite Paul Sabatier
118, route do Narbonne
31062 Toulouse Cedex, France

Brian R. Gaines
Knowledge Sciences Institute
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, Canada T2N 1N4

Ute Gappa
Karlsruhe University
Institute of Logic
PO Box 6980, D-7500 Karlsruhe 1
West Germany

Ashok Goel
School of Information and Computer Science
Georgia Institute of Technology

Fernando Gomez
Department of Computer Science
University of Central Florida
Box 25000
Orlando, Florida 32816 USA

Linda Green
Information Systems Laboratory
Advanced Technology Center, 7L-64
Boeing Computer Services
Box 24346
Seattle, WA 98124 USA

Tom Gruber
Knowledge Systems Laboratory
Department of Computer Science
Stanford University
701 Welch Road, Building C
Palo Alto, CA 94304, USA

Max Henrion
Rockwell Science Center
444 High St.
Palo Alto, CA 94301

Toshikazu Hirobe
Automation Technolgy Center
Chiyoda Corportion
12-1, Tsurumichuo 2-chome,
Tsurumi-ku

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

Chris Jacobson
Computer Research Laboratory
Tektronix Laboratories
Tektronix, Inc.
Box 500 MS 50-662
Beaverton, OR 97077 USA

Jay F. Jnamaker, Jr.
Department of Management Information Systems
College of Business and Public Administration
University of Arizona
Tucson, Arizona 85721 USA

Todd R. Johnson
Laboratory for Artifical Inteligence Research
Department of Computer and Information Science
2036 Neil Avenue Mall
Columbus, OH 43210-1277

William P. Jones
The Applied Artificial Intelligence Center
Arthur D. Little, Inc.
20 Acorn Park
Cambridge, MA 02140 USA

Robert L. Joseph
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213 USA

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

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

Ichirou Koshijima
Automation Technolgy Center
Chiyoda Corportion
12-1, Tsurumichuo 2-chome,
Tsurumi-ku

Juliana S. Lancaster
The BDM Corporation
1300-North17th St., Suite 950
Arlington, VA 22209 USA

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

Ting-peng Liang
Department of Accountancy
University of Illinois at Urbana-Champaign
1206 South Sixth Street
Champaign, Illinois 61820 USA

Yihwa Irene Liou
Information and Quantitative Sciences Dept
Robert G. Merrick School of Business
University of Baltimore
Baltimore, MD 21201 USA

Jonathan I. Maletic
Dept. of Computer Science
538 Mackenzie Hall
Wayne State University
Detroit, Michigan 48202 USA

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

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

Karen L. McGraw
Cognitive Technologies
2315-B Forest Drive, #9
Annapolis, MD 21401 USA

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

R. S. Michalski
Artificial Intelligence Center
George Mason University
Fairfax, VA 22030 USA

Hiroshi Motoda
Advanced Research Laboratory
Hitachi, Ltd
1-280 Higashi-Koigakubo
Kokubunji, Tokyo 185 Japan

Humio Murakami
Nippon Felt Co., Ltd
88, Haramarumo, Kononsushi
Saitama 365, Japan

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

Kazuo Niida
Automation Technolgy Center
Chiyoda Corportion
12-1, Tsurumichuo 2-chome,
Tsurumi-ku

Frederick E. Petry
Tulane University

Stephen E. Porvin
Dept. of Computer Science
538 Mackenzie Hall
Wayne State University
Detroit, Michigan 48202 USA

Jukka Rantanen
Laboratory for Information
Processing
Technical Research Centre of
Finland (VTT)
Lehtisaarentie 2A
SF-00340 Helsinki, Finland

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

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

Robert G. Reynolds
Dept. of Computer Science
538 Mackenzie Hall
Wayne State University
Detroit, Michigan 48202 USA

Marie-Helene Riviere
Institut de Recherche en Informatique
de Toulouse
Universite Paul Sabatier
118, route do Narbonne
31062 Toulouse Cedex, France

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

Guus Schreiber ,
Department of Social Science
Informatics
University of Amsterdam
Herengracht 196
NL-1016 BS Amsterdam, The
Netherlands

Valerie L. Shalin
Honeywell Systems and Research
Center
3660 Technology Dr. MN65-2500
Minneapolis, MN 55418 USA

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

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

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

Wang Shenkang
Department of Computer Science
University of Ottawa
34 George Glinski
Ottawa, Ontario, Canada K1N 6N5

Douglas Skuce
Department of Computer Science
University of Ottawa
34 George Glinski
Ottawa, Ontario, Canada K1N 6N5

Jean-Luc Soubie
Institut de Recherche en Informatique
de Toulouse
Universite Paul Sabatier
118, route do Narbonne
31062 Toulouse Cedex, France

Jeffrey Stout
Information Systems Laboratory
Advanced Technology Center, 7L-64
Boeing Computer Services
Box 24346
Seattle, WA 98124 USA

Masaki Suwa
Advanced Research Laboratory
Hitachi, Ltd
1-280 Higashi-Koigakubo
Kokubunji, Tokyo 185 Japan

Hirokazu Taki
ICOT
Institute for New Generation
Computer Technology
4-28, Mita 1-Chome, Minato-Ku
Tokyo 108 Japan

Jun Tani
Automation Technolgy Center
Chiyoda Corportion
12-1, Tsurumichuo 2-chome,
Tsurumi-ku

E. Sue Weber
Department of Management
Information Systems
College of Business and Public
Administration
University of Arizona
Tucson, Arizona 85721 USA

Bob Wielinga
Department of Social Science
Informatics
University of Amsterdam
Herengracht 196
NL-1016 BS Amsterdam, The
Netherlands

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

J. Zhang
Artificial Intelligence Center
George Mason University
Fairfax, VA 22030 USA

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

KNOWLEDGE FOR DESIGN

T. R. Addis

Knowledge Systems Group
Computer Science Department
University of Reading, UK.

Abstract

Knowledge-based systems provide intellectual assistance through the manipulation of symbols. The theoretical basis for this manipulation is derived from the logic research programme which restricts the creation of models to predefined symbols, a semantics based upon truth and valid inference. However, it is made clear that the design of any artefact, including a knowledge-based system, will require one or more representations that emerge from the creation of new meanings to symbols, a semantics based upon experience and non-valid inference. It is a representation that models an abstraction of the world and that is abduced from a theory. It is proposed that there should be a parallel research programme to that of Logic. A programme that investigates the human use of theories, models and artefacts in design and discovery.

1. KNOWLEDGE-BASED SYSTEMS

Knowledge-based systems are a product of Artificial Intelligence research and such systems have taken on many roles as people come to terms with the idea of manipulating "knowledge". In the first instance the knowledge-based system was considered by many people to be a direct replacement of an expert. The "Turing test" was often evoked as a paradigm for a knowledge-based or expert system. This paradigm is that where given the limited means of communication with a computer then it should **not** be possible to tell the difference between the knowledge-based system and an expert (hence the term expert system). Designers of such systems had high expectations that have never really been fulfilled.

The Turing test has always been a misconceived approach to a specification of the objectives of Artificial Intelligence since it has the appearance of a clear definition in that it uses equivalence; but this notion of equivalence does not include the most important element. This element is the exact manner in which a user of a knowledge-based system would attempt to make the distinction between a person and a program or to determine the criteria to be satisfied that a person and a program are equivalent. It is within this set of actions or this specification of the criteria that the real difficulties lie.

The analogue of knowledge was often considered (although I believe only half seriously by most practitioners) as a sort of substance that was extractable (mined) from experts. The extraction process (knowledge acquisition) is accomplished by interviews with the expert where every utterance, action and gesture (if possible) is recorded on audio and/or video tape. The script is then analysed (the ore refined) by a knowledge engineer sometimes with the help of an "induction" program. The induction program will identify (with respect to the tasks to be performed by the expert) the

information-carrying statements or actions and reject redundant statements or actions.

The result of knowledge acquisition would be a model of the expert in the form of a representation that was at least indirectly convertible to a computer implementation. The target representation will depend upon the knowledge-based system used. Many systems are variants of a "production rule" system which presumes the existence of a deductive inference unit that can work under the guidance of some simple heuristics to infer an expert's actions from the model. The model of the expert is given in the form of "if ... then ... " rules. Many of these expert system models may be fine-tuned with the aid of weightings (sometimes called certainty factors or probabilities) that are assigned to each of the rules.

These models can never be complete since the systems that use them rarely have the capability to interact directly with the environment and even those that do will be deprived of many of the refinements of the human senses. The models have to rely on a human agent (the user) who must be able to interpret the meaning of the words generated by the model through the system to instruct the user in an exploration of and the actions to be performed in the task domain (the expert's field). Thus a less ambitious view of a knowledge-based system is as that of an aid or intellectual assistant to a skilled user (Addis, 1985). The exact position of a knowledge-based system in the scheme of activities is shown in figure 1.

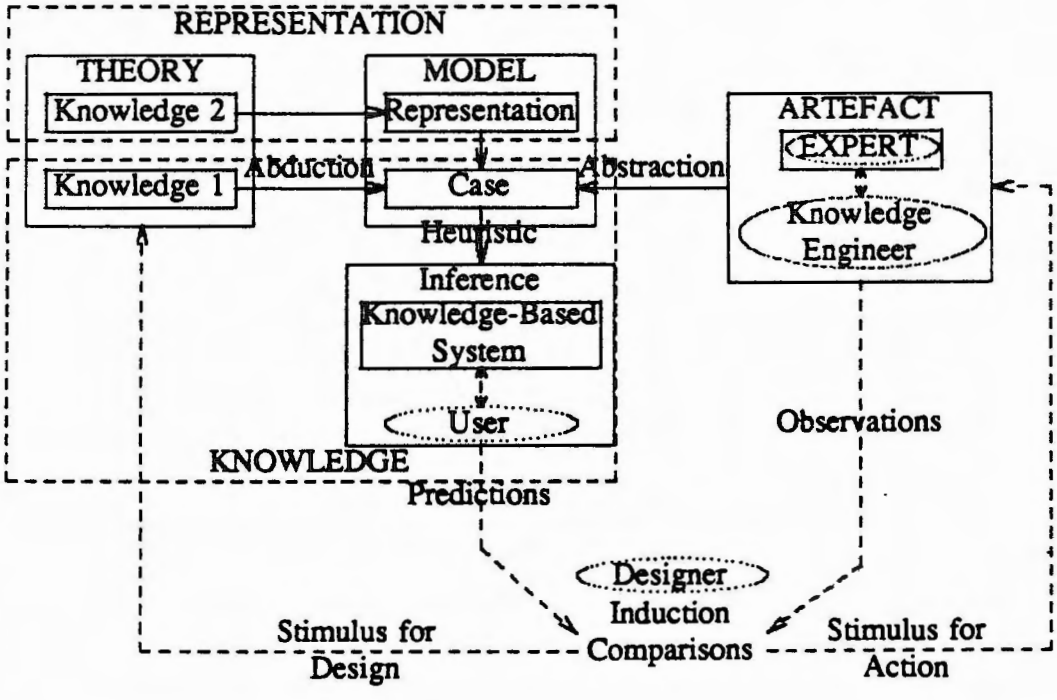


Figure 1. The role of a knowledge-based system with respect to a theory.

The model is referred to as the "knowledge of the expert". The model is derived from the knowledge acquisition results provided by the knowledge engineer and these results are interpreted by the designer in terms of the theory (Figure 1; Knowledge 1). The process of deriving the model from the theory depends upon abduction and abstraction (formerly called type 2 and type 1 abduction respectively - Addis 1987).

The model is a "case" (an element of the extension) of the theory of knowledge given in terms of some representation scheme (Knowledge 2). The model is driven by deductive inference and controlled by heuristics (Abduction type 3) towards a chosen goal. Induction uses the results of the model and observations to either confirm (justify) the model or to stimulate further action and design.

Knowledge-based systems provide possible intellectual assistance through the animation and automation of representations (vehicles of thought). In order to build better aids to thought we need a better grasp of what is involved in the human use of representations (so we can automate aspects of that use). The account which is currently in AI is based upon problem-solving as portrayed by Newell and Simon (Newell and Simon 1956). This theory (of knowledge) is a derivative of a human problem-solving model, an example (a case) of which was first implemented as the Logic Theory Machine (LTM). LTM was later extended to a General Problem Solver (GPS). The theory was proposed in order to describe the general behaviour of humans reporting on their thought processes as they were solving problems (Newell, A. and Simon, H. A. 1973).

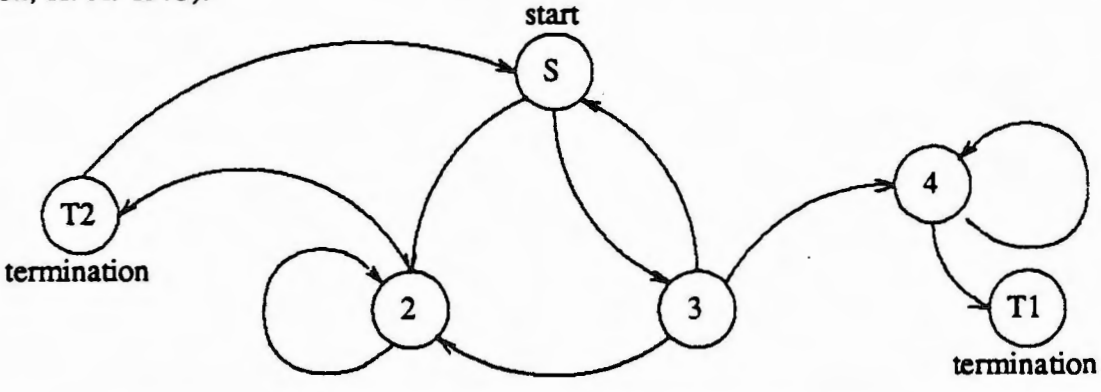


Figure 2. States and transformations in a problem space.

In essence, the theory states that problem solving is a process of exploring a "problem space" for a solution. A problem space can be considered as a directed graph (see Figure 2). The nodes of the graph represent the different possible states of a problem. There is usually a single start state that indicates the current situation and a set of termination states that indicate the desired result. The arrows that go from node to node represent the actions that are available to an agent in that state and these actions transform the problem from one state to the next. Problem solving is the discovery of a path from the start state to a termination state. The solution is the sequence of transformations (arcs) that make up the path.

The theory has its foundations in utility theory (Luce and Raiffa, 1957). Utility theory is a means through which choices can be assessed (their utility) and decisions can be made. The utility of a state in Newell and Simon's version of decision theory is a measure of the proximity that a state (and hence the choice that leads to that state) has to a solution combined with an estimate of the cost in the solution path (transformation sequence). It is this utility measure that forms the basis of a representation of the heuristic knowledge since it is the utility that guides the inference system to select a route to a solution.

The computer model of a general problem solver is expected to consist of three elements that are related to different kinds of case knowledge:

- a data representation that describes a set of problem states and has the potential to describe all the problem states (the abstraction of the problem),
- a set of data updates that describes how a given state can be transformed to a new state (the deductive system),
- a set of heuristics that provides guidance to a search algorithm through the problem space (the heuristic knowledge).

These elements may be represented in a wide variety of forms depending upon the knowledge representation scheme used (eg clauses, rules). Many AI programs will have these three elements made explicit in their design even though they may have been designed through a different theoretical framework (eg. object orientated programming, semantic nets, frames, case systems, etc). Most of the alternative current AI theories (of knowledge) are concerned mainly with representation.

The appeal of Newell and Simon's theory is that the states of the problem can be represented as propositions. These propositions are formal representations of natural sentences that are written thus:

All(x) Elephant(x) -> Colour(Grey,x)

Exists(y) Name(Dumbo,y) and Elephant(y)

and the transformation from one state to the next is the application of deductive inference to selected propositions. The deductive inference step is an extension of Modus Ponens called Resolution. The only requirement is that the propositions must be normalised into clauses; a process that can be done automatically. This process is appealing because it demonstrates "human thinking" as proposed by logicians. Further, it defines the start of an important research programme into modelling human cognition that has deep foundations in Mathematics and the formal traditions of Science. The programme can draw upon a long history of development that includes work from many of the best thinkers of the last two millennia. This research programme has been pursued vigorously by many centres in the UK such as Imperial College (London) and Edinburgh University.

2. THE ROLE OF A MODEL

Newell and Simon's problem-space is defined by means of linguistic (propositional) representations and problems are solved by manipulating linguistic representations. This has stimulated the development of animated and automated sentence manipulators (the logic research programme). But scientific and engineering thinking uses other vehicles, viz. models, which should not be regarded as reducible to sets of sentences.

As an illustration of what is outside the philosophy of the logic research programme, consider the process of designing a simple slide as shown in Figure 3. Such a slide may form part of the transportation system for a product in a factory. This product will have specific dimensions, weight and composition. After the drawing of a sketch that shows the spatial relationship of the components of the slide, it is required to determine the acceleration of the product down the slide so that the velocity may be

calculated at its point of reception.

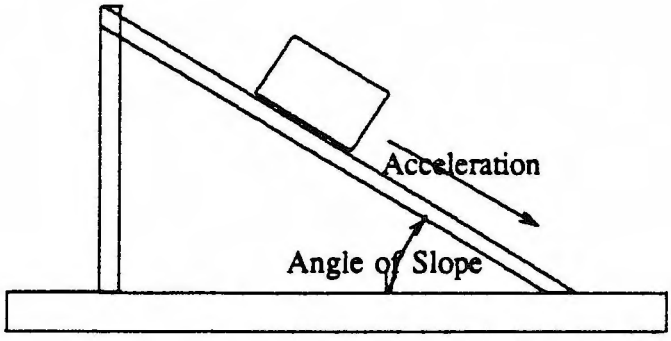


Figure 3. A spatial representation of a slide.

Reference is made to the theory of Dynamics. Some aspects of this theory† may be presented as a series of statements thus:

T1. Every Body travels in a straight line unless a force acts upon it.

T2. Momentum is the product of mass and velocity.

T3. Force is the rate of change of momentum.

T4. For every action there is an equal and opposite reaction.

T5. Gravity is an acceleration caused by the mutual attraction of mass.

T6. Weight is a force due to gravity.

T7. Forces (and hence velocities) will add as vectors.

From this theory a dynamic model may be constructed using mathematics. In this case a first attempt will appear as:

M1. A Body of mass M on a slide.

M2. The slide is at angle θ with respect to the horizontal.

M3. The weight of the Body is $W = M*G$ where G is the acceleration due to gravity.

M4. The force down the slide $Fs = W*\text{Sin}(\theta)$.

The slide is constructed and it is found that the actual acceleration is much less than that predicted. After checking the calculations, reference is then made back to the Theory (see Figure 4). The Theory restricts the set of possible explanations for this discrepancy and guides the designer to create a better model. In this case, the theory limits the set of possible proposals to the existence of another force acting on the Body. This force then "explains" the discrepancy between the model and the observations; this force is friction.

The notion of friction does not come directly from the theory but is an interpretation of both experience (ie. there is a recognizable feeling of resistance when moving the Body on the slide which becomes greater with increased pressure between the surfaces) and the Theory (ie. only forces can influence the motion of a Body). The

† A theory in this paper does not necessarily mean a formal theory. Any set of statements that forms some coherent description of the world that can be used to "render facts likely" (Peirce, 1934) will be considered a theory. A theory is a set of propositions that reduces the uncertainty in the world for an agent.

process that generates the insight (that these disparate experiences and concepts should be amalgamated into a simple causal element identified as friction) is called "abductive" inference. An important component of abduction is the element of contact by an agent with the world. The Model is modified thus:

M4'. The force down the slide Fs' = W*[Sin(θ) - K*Cos(θ)] where K is the coefficient of friction for the materials in contact.

The coefficient of friction K is a concept that has evolved from the need to adjust the model to fit the observations (see Figure 4). However, the notion of friction does not and cannot emerge from only the manipulation of the sentences that make up the theory.

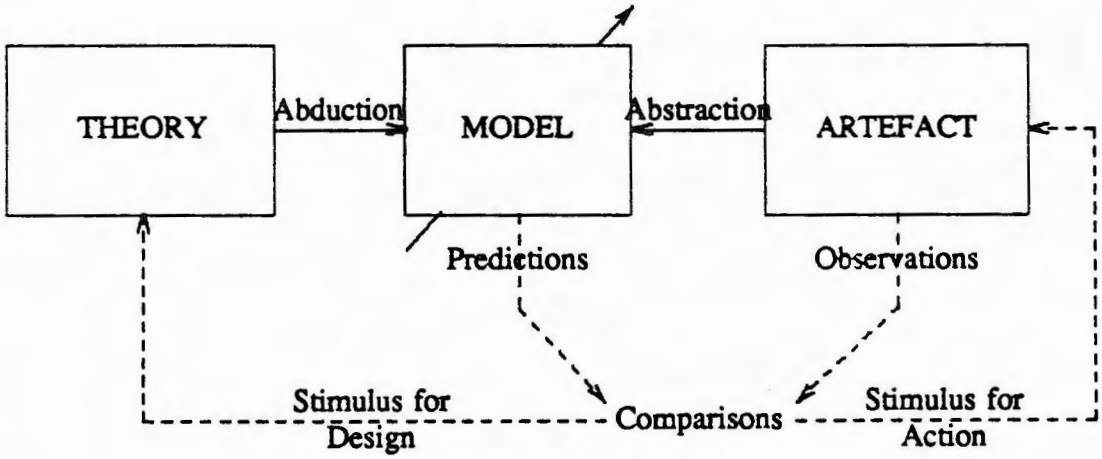


Figure 4. A cycle in the design process.

A precise relationship between a theory and a model cannot be easily defined as they both represent a range of possible interpretations and share many of the same properties (cf Aris, 1978). There is a continuum of theories/models ranging from the non-specific (most general) to the identification of unique cases. The relationship between a theory and a model is that a model will be associated with a more specific situation (a case) than a theory. The complete range of models derivable from a theory will be called the "extension" of the theory. The theory from which a model is derived is called its "intension".

Figure 5 illustrates the model as the end product of several stages of **abduction**, the model emerges from the potentially infinite set of possibilities that may be presumed to exist within the abstract world of human imagination. However, the model is also the end product of several stages of **abstraction** of the potential distinctions in the empirical world. The model captures only a subset of the possible features of an artefact (ie it does not, in the above case (example), show the colour, texture, smell or structure of the materials); it is a subset that is coherent and serves a particular purpose. A model and a theory both together encapsulate an understanding of the world that is the result of purpose and experience. A model and its associated theory will be called "declarative knowledge" if the model relates to at least one feasible artefact (ie can be constructed). What is feasible and the methods used to make such a judgement (such as the comparison of predictions with observations) are referred to as

"inductive" inference.

In the case of the slide the mathematical model will describe the active forces of interest provided the right combination of calculations is applied. The knowledge of how to perform a particular chosen calculation correctly is "deductive" inference. Mathematical calculations are deductive in the sense that they are "valid" operations and truth preserving. The understanding of how to perform the three forms of inference; abduction, deduction and induction will be called "inferential knowledge".

The right combination of calculations is selected by the mathematician given the purpose of the model. This understanding of how to use the model to make appropriate predictions will be called "heuristic knowledge". Heuristic knowledge may be considered to be the result of another form of inference since it requires a similar kind of insight observed in abduction; the perception of the significance of certain particulars.

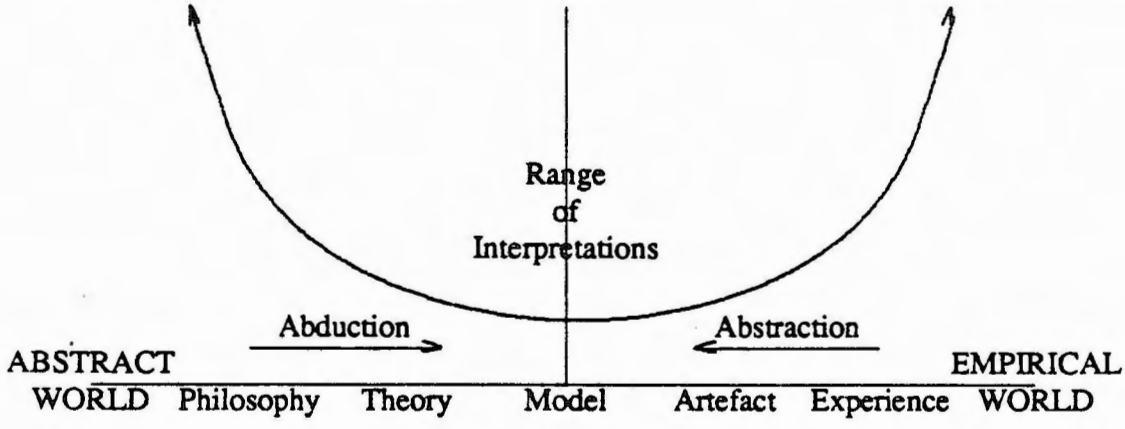


Figure 5. The Model as a point of contact between Theory and Artefact.

The design of this slide (the artefact) has currently utilised two models; the model of forces represented in mathematical terms and the model of spatial relationships that uses a scaled projection of the slide on to 2-dimensions. The two models are two different abstractions from experience and two different abductions from theory. The models are related in that there is a clear mapping between points of contact. In this case the points of contact are the angle of the slope and the direction of the forces acting on the Body†. Other design tasks, such as the construction of VLSI chips, will use nine or ten different models to represent different abstractions of the same object. Each of the models is related to some theory that uses a generalisation of the characteristics to be controlled and formed in the design. A theory is only useful if it provides this control. The control is incorporated in the constraints (lawful behaviour) and in the procedures for deriving the consequences of any design decision within the domain of the theory. Hence the laws of electronics (eg. Ohm's Law) provide a means through which particular circuits (electronic models) may have predictable performance.

Related to each model type are one or more preferred representations and a means of deducing measurable features of the artefact. Before the invention of

† However, the triangle of forces is not shown.

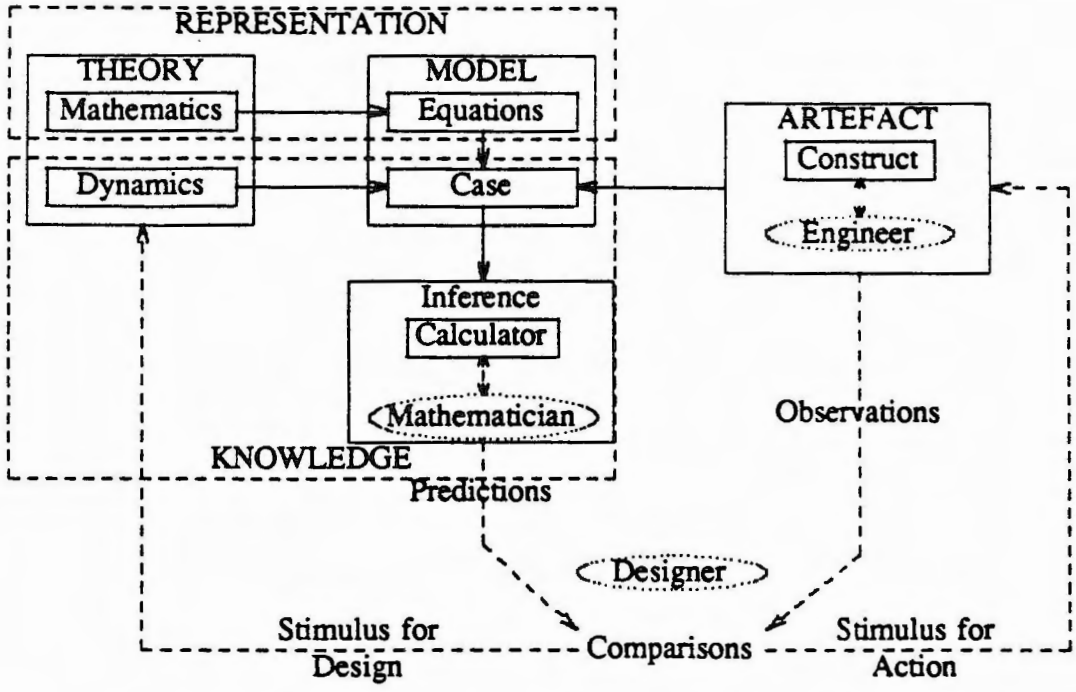


Figure 6. The representation of knowledge and the skills for design.

Cartesian Projection as a method of depicting a normalised view of objects, the only means of representing spatial relationships was through sketches. Sketches (or perspective drafts/drawings) do not allow the possibility of extracting measurements or making detailed predictions of the distance between any two points on the object. Sketches do show certain ordinal relationships such as next, above, below, inside and meshing. However, the Cartesian projections, with the aid of a ruler and in some cases a little geometry, will provide not only this ordinal information but will also give a prediction of the distance between any two points.

Figure 6 shows details of the elements involved for one of the models in designing the slide. The theory involves both the representation scheme (mathematics) and the generalisation of a particular aspect of the world (dynamics). The theory is made explicit by the model in the form of equations (case). The model is influenced by both the theory and the artefact. The model is interrogated through calculations by a mathematician (usually the same person as the designer) who may use a calculator or tables. The engineer must be able to interface with the artefact (construct) through measurements (eg. the angle of slope and the acceleration) in order to provide observations that can be compared with the predictions of the designer; the artefact must engage the model. The mathematician, engineer and designer indicate the skills (tacit knowledge) required to progress a simple design.

Although the model is the main component to be altered in the design process, the effectiveness of both the theory and the artefact are continually under review. The theory will be modified if the models can rarely be made adequate to reflect the artefacts and an artefact will be reconsidered in the light of predicted performance and the achievement of the purpose. Progress can only be made if there exist both the

beginnings of a theory and the inklings of an artefact from which a case model can be created. However, to modify a theory would seem to require the existence of a meta-theory which has the same relationship to the theory as that which the theory has to the model. Model, theory and meta-theory will be referred to as different "levels" of knowledge. A theory is abduced from a meta-theory and supported through induction from experience with its models.

3. THE LIMITS OF LOGIC

The logic research programme, after some initial success, seems to have reached a point where progress has slowed. There are several reasons for this; technical and fundamental.

The "technical" reasons are to do with the excessive expansion of the problem space when faced with a small increase in problem scale (known as the combinatorial explosion). The attainment of better heuristic knowledge and methods of assessing the state utility is a major objective in this research programme. However, there is currently no general theory that explains heuristics nor describes how such knowledge may arise. Each case must be dealt with on its own merits where the heuristic component is supplied by the designer.

Techniques for the system to automatically extend the model as the problem space is explored have had some success in reducing the search space for some problems (eg. Truth Maintenance Systems). However, in many cases these elaborate techniques have had only marginal effects. In the end, it usually falls upon the designer of the system to incorporate as much domain knowledge as possible in the heuristic component.

An alternative approach to the search space problem is to use more powerful computers and in particular to use parallel processing. It would seem natural to use parallel exploration of a divergent problem space but the amount of improvement depends upon the structure of the problem space. The control and inter-communication of the parallel processes for optimum effect is currently a research topic.

The "fundamental" reasons for the slowing up of progress call into question many of the assumptions on which logic and formal systems are based. It is clear from the description of the general structure of Artificial Intelligence programs that: firstly the concept of truth never enters into the issue of design except as a marker to show valid deduction and secondly there are several elements that still have to be provided by the designer, knowledge engineer and user. These are:

- The heuristic knowledge and purpose for which the solution is required (the heuristic component).
- The range of interpretations of the symbols and structure in the model (the abductive component).
- The association of the distinctions in the world with the primitives for each case (the abstractive component).
- The creation of a model from a theory and its subsequent modification in the light of experience (the abductive and inductive component).

These are similar or associated reasons to those reasons why Wittgenstein explored an alternative view to his Tractatus and why Peirce concerned himself with semiotics and abductive inference. Part of the problem can be illustrated by a re-drawing of Figure 3 as shown in Figure 7.

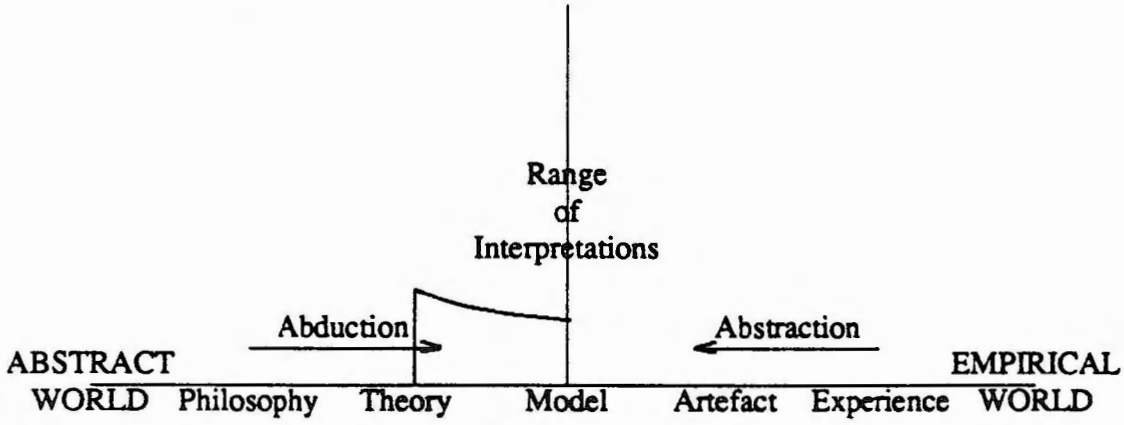


Figure 7. The boundaries of first and second order logic.

Figure 7. suggests that the boundaries of first order and second order logic are restricted to the abstract world and limited to the relationships between theory and its models. Thus, the range of interpretations is not open-ended†. The mapping of theory to model is a second order issue since the theory is used to generate new predicates. In the slide example the new predicate Friction(x,f) was implied where x is a body and f the frictional force. However, the mechanism through which this generation of predicates occurs is not fully understood; second order logic is incomplete. It is for this reason that the counter argument that might be given by logicians concerning the generalization of logic to include any number of stages of abstraction cannot be fully justified; the complete formal step between levels of knowledge has not yet been achieved. However, there are two-level "abductive" programs that incorporate both theory and model based on something like a simple second-order logic (eg. Langley et al, 1987). Nevertheless, the engagement of the model with the world still requires human action except in some very primitive cases (eg thermostats and other control mechanisms).

The argument for continuing the logic research programme is strong and it is not the purpose of this paper to suggest otherwise. What is being proposed is that alongside this research programme is required a parallel research programme which includes the human agent and a programme which attempts to provide a richer structure to the knowledge description; a structure that can cope with a formal description of heuristics, interpretation, abduction and induction, a structure that may even be able to define intelligence unambiguously.

The benefit is not a proposal to develop automated models - that's already commonplace - but to provide automated aids to the process of moving between artefact, model and theory. The relationship between artefact and model, and model and theory

† However, there may be an infinite set of interpretations within each range. The vertical axis in Figure 5 and in Figure 7 is an ordinal measure of transfinite numbers.

is one of generalization and the major limitation of the logic research programme is that by confining its attention to linguistic representations it cannot adequately encompass the full process by which humans move between the particular and the general. It is not merely the incompleteness (logician's sense) of second-order logic which stands in the way, it is the incompleteness (ordinary sense) of the representations which logician's use. Here is where there is room to run a research programme alongside the logic programme (Addis, 1989).

4. ACKNOWLEDGEMENTS

I would like to thank Dr David Gooding of the University of Bath for drawing my attention to the importance of "activity" in the process of discovery and for commenting on this paper. I would also like to thank Dr Jim Tiles who has helped me form a coherent argument from the essential elements presented here and Garfield Dean for his many useful observations. However, the interpretation of these discussions and any errors are mine alone.

5. REFERENCES

1 Addis T. R., (1985) *Designing Knowledge-Based Systems*, Kogan Page, London/Prentice-Hall.

2 Addis T. R., (1987) *A Framework for Knowledge Elicitation*, The First European Workshop on Knowledge Acquisition for Knowledge-Based Systems, September, Reading University.

3 Addis T. R., (1989) *The Science of Knowledge: A Research Programme for Knowledge Engineering*, The Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems, July, Paris.

4 Aris R., (1978) *Mathematical Modelling Techniques.* Research Notes in Mathematics 24, Pitman, London.

5 Langley P., Simon H. A., Bradshaw G. L. & Zytkow J. M., (1987) *Scientific Discovery: Computational Exploration of the Creative Process*, MIT press.

6 Luce R. D. & Raiffa H., (1957) *Games and Decisions: Introduction and Critical Survey.* John Wiley & Sons Ltd.

7 Newell A. & Simon H. A., (1956) *The Logic Theory Machine.* IRE Transactions on Information Theory. Sept.

8 Newell A. & Simon H. A., (1973) *Human Problem Solving.* Englewood Cliffs, N. J: Prentice-Hall.

9 Peirce C. S., (1934) **Scientific Method** in *Collected papers of C. S. Peirce*, Vol VII, ed A. W. Burks.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A MEDIATING REPRESENTATION TO ASSIST KNOWLEDGE ACQUISITION WITH MACAO[1]

-DRAFT-

Authors: Nathalie AUSSENAC, Jean FRONTIN, Marie-Hélène RIVIERE, Jean-Luc SOUBIE (Projet ARAMIIHS)

Address:
Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier
118, route de Narbonne
31062 Toulouse Cedex
Phone: 61.55.63.20

Centre MATRA-Espace de Toulouse
Phone: 61.39.68.12

Keywords :
Knowledge acquisition, Mediating representation, expert psychological model, knowledge graph display.

Abstract :[2]

Knowledge acquisition from a human expert remains a major problem in expert systems design. A set of method, that are all the more relievant as they consider the nature of the expertise, are now available. Our experience of a multi-expert system development has led us to define a common methodology for the various expertise domains: MACAO. It assists the knowledge engineer in his most difficult tasks, with the help of a software tool.

In this paper, we stress the specifications of the knowledge representation in the software. It requires a close definition with the double aim to help in the knowledge collecting process as well as the validation of the obtained data.

Mediating representations seem to meet these properties. They permit the expert to visualize the collected knowledge and to complete or correct it. This kind of representation is the one selected for MACAO, where an internal and an external formalizations are combined. The internal one integrates the concepts of an expert cognitive model. The external representation corresponds to a hierarchy of graphs, the semantics of which is given by the internal structures.

We expose our experience of realization of such a software and we present the first prototype screens as examples.

---

[1]This research is jointly developped by a group of MATRA-espace engineers and a team from the IRIT institute. It is partially funded by the "Conseil Régional de Midi-Pyrénées" and the PIRTTEM of the CNRS

[2]This issue of the paper is a first version to be consistently improved with the help of a native english speaker.

1 INTRODUCTION

Because the problem of knowledge elicitation and analysis remains one of the major hindrances to a wider development of knowledge based systems, an increasing number of methologies and tools are now available for knowledge engineers (ETS [BOOSE,86], KADS [WIELINGA,84], OPAL [MUSEN,88], MORE [KAHN,85],...). Many of those tools offer sophisticated knowledge editors, and stand as good supports for domain knowledge encoding and "book-keeping". They also provide the knowledge engineer with partially automated techniques for high-level analysis and knowledge structuration, after it has been conceptualized or modelled.

Except the tools whose efficiency has been proved by numerous applications, most of them refer to rather primitive methods. As a matter of fact, few methods consider the acquisition process as a whole, from "early conceptualization to validation", as stressed by N. Johnson in [JOHNSON,88] . For instance, very few tools enable experts to enter directly their knowledge without learning the use of computational representations.

In this paper, we first argue that an effective tool-based method for expertise transfer should meet the following requirements:

- take into account the nature of expertise by refering to a conceptual or cognitive model;
- integrate in a methodology psychological techniques for knowledge retrieval and structuration;
- priviledge the expert direct expression without constraining him into putting his knowledge into a rigid formalism;
- provide the expert with expressive and explicit displaying devices, so that he could validate, update and test the formalized knowledge.

To this end, when defining the MACAO methodology, we have choosed to integrate an expert cognitive model, and to support the knowledge engineer tasks with an acquisition tool [AUSSENAC,88b]. Moreover, we believe that knowledge representation plays an important part in the tool success in acheiving those aims. This is why an acquisition tool with a "custom-tailored" knowledge representation defined in keeping with the method and the model, seems a good way to improve the software aid power.

This paper shortly recalls some of the model caracteristical features as well as the method major stages. It presents the usual kinds of knowledge representations in knowledge acquisition tools, focusing on the way structuration is carried out in each case. The interest of "mediating representations" is outlined [JOHNSON,88]. Then, the internal and external representation formalisms in MACAO are described. The help they provide during the conceptualization, analysis and validation steps is pointed out from an example.

2 THE MACAO METHODOLOGY FOR KNOWLEDGE ACQUISITION

2.1 The main features of the expert cognitive model

2.1.1 Conceptual and cognitive models

When knowledge engineers build expert systems, their paper consists in elicitating experts' knowledge, then structure and formalize it as reliablely as possible. They aim at the target system being able to simulate the experts' problem solving behaviour. While they carry out this process, the knowledge engineers form implicit or even empirical models of how the experts solve problems. For instance, many of them assume that all the expert's knowledge is available in his discourse during interviews, whatever the expertise may be.

Actually, experts refer to their own mental models, which the knowledge engineer should be acquainted of. Knowing the way the expertise is structured and processed according to the kind of problems to be solved, seems to provide help when elicitating it. Moreover, some results from psychology, and mainly cognitive models definitions, bring light on the experts' problem solving processes.

Many psychologists' research, such as Berry's [BERRY,87] or Rietman and Olson's [REITMAN,87], as well as Artificial Intelligence ones [GAINES,88], tend to prove that the use of such models may overcome most of the knowledge retrieval difficulties because it helps to enhance various aspects of the expert's behaviour and select a set of appropriate elicitation techniques.

Wielinga's models of expertise deal with a similar matter. Those models lay at different levels and describe the abstract structures that form the expert's domain knowledge. They differ from cognitive models because the expert takes part in their elaboration, and because they include domain particularities [WIELINGA,84].

Problem or domain models are kinds of conceptual models that Musen defines as "models of the problems and the way experts solve them". These models are "the semantic framework shared by the system users and developpers that permits them to communicate" [MUSEN,88]. He proposes a classification into symbolic models (or computational ones), method-oriented models (which may be implicit or explicit descriptions of the problem solving behaviour), and task-based models.

A conceptual model stands indeed as a good means to derive from the expertise a shared semantics for experts and knowledge engineers. So we planned to refer to a cognitive model in order to apprehend better the expertise first, and then define the main concepts of a task-based conceptual model.

2.1.2 Some aspects of the expert model

The model we use studies the structure of knowledge through its evocation, storage and evolution. It assumes that, when an expert solves a problem, he accedes to his knowledge by means of particuliar information selected in the environment: the objects of the expertise. These objects not only have physical caracteristics, they also correspond to a mental model for the expert, labelled **functional unit.**

To depict the storage of the expert's knowledge, the model relies on different schemes and concepts.

Empirical schemes , located on a less abstract level, describe the knowledge used to solve and carry out some parts of problems: we call them procedures. They consist of sets of functional units instances and values. So solving a problem requires the calling of several "procedures".

Conceptual schemes , located on an abstract level, describe the expert's knowledge concerning his strategies and the choice of convenient procedures: they are a kind of meta-knowledge. They describe conceptual relations and types of concepts.

The problems the resolution requires a similar set of knowledge at both levels, belong to the same "category of problem". Some particular objects at the higher level guide the expert in selecting the procedures to be applied when solving a given problem: we call them the "determinants". All these concepts are the base for the analysis in the third part of the methodology.

Thus, expertise transfer amounts to put forward the domain functional units, their relations and structuration according to the category of problems. This structuration is depicted regarding two aspects, a static and a dynamic one (explained later in paragraph 3).

2.2 The MACAO method stages

The MACAO methodology is based on the observation of the expert at work. The whole acquisition process is supported by a software tool. The method allows the construction of a formalized knowledge set, according to a mediating representation selected in keeping with the cognitive model specifications. It supposes that the persons involved in the expert system development have been informed enough about the project. It is also assumed that a complete feasability study has been carried out before [BENCHIMOL,86].

The knowledge analysis takes place at different levels according to the following four main stages:

1. **Initialization**
   This first stage is based on a sequence of dialogs between the expert and the knowledge engineer, combined with the observation of the expert at work. They specify the user's environment, the categories of problem to be solved and the main features of the expertise. This set of information forms the software initialization nucleus.

2. **Protocols elicitation**
   The second stage is carried out with the prospect of a protocole analysis. The expert is invited to solve several problems of the categories he has mentioned. He is placed in a situation as close as possible to reality. Meanwhile, a track of his behaviour is kept by two means. Fisrt, the expert explains what he is doing by "thinking aloud" or giving the additional information that the knowledge engineer may ask him for. The dialogs are recorded. Then, he indicates sequentially all the data he needs and enters in the software their name (forming the problem "glossary"), as well as their generic label (belonging to the domain lexicon). For example, when an operator oversees a bus network, and means that a bus is in advance, he writes "advance" in the glossary associated with "disturbance" in the lexicon.

   These first two stages correspond to knowledge identification and early conceptualization.

3. **Knowledge conceptualization, analysis and formalization**
During this next stage, the knowledge engineer analyses the information retrieved through the frame suggested by the model. After concepts and relations are formalized into internal structures, the tool helps the knowledge engineer to compare systematically the sequence of structures involved in the resolution of each problem by studying similarities and differences. The tool warns him of inconsistent formalizations or may suggest to gather functional units into procedures. This analysis leads to the construction of several knowledge graphs, corresponding to each problem solving or describing each category. It concerns both the static and dynamic knowledge entities. Because of their major role in this stage, the internal and external (graphs) representations will be described in part 4.2.

4. **Confrontation with the expert, knowledge validation**
As the expert is the only one to judge the accuracy of the retreived and structured knowledge, he validates it directly in this last stage. He may correct the general knowledge graph, or go through the graph hierarchy in order to control the ones corresponding to categories or to problems. A set of displaying and updating tools is at his disposal for an easy knowledge browsing. For instance, consistency is automatically checked. The knowledge engineer may guide him or justify the way he has formalized. Validation goes on until the expert is satisfied with the whole graph hierarchy.

3 KNOWLEDGE REPRESENTATION AS A SUPPORT TO STRUCTURATION

3.1 Knowledge representation: a paradigme to classify acquisition tools

In the existing knowledge acquisition tools, the problem of knowledge representation is tackled under two points of view. Either the process is oriented by the target development environment. Then, the elicitated pieces of expertise are represented directly in the knowledge base formalism. Or the tool has its own structures to store and analyse the knowledge. In that case, an ultimate translation is required to get the knowledge base.

The available tools also differ in the way they edit the knowledge once it has been structured or during its structuration. The range of the editors abilities goes from "rought" but operational code editions to sophisticated visualizations with poor semantics. As a matter of fact, some systems like KREME [ABRETT,86] rely essentially on the editor help to guide the knowledge classification. We will call such tools "visual representation" based tools, whereas the ones of the first kind are "knowledge-base oriented", and the second ones are based on an "intermediate representation".

3.1.1 Computational languages and acquisition tools

The tools of this first category often offer to the user a powerful and structured language. But their lack of flexibility may limit or distort the expert expression. Moreover, if the knowledge edition format is too close to the code, it is difficult for the expert to understand it unless he is familiar with programming languages. So even if those tools provide the knowledge engineers with efficient help, they cannot be easily used by experts.

This issue was particularly acute in first generation expert systems where knowledge was represented exclusively with production rules. For instance, TEIRESIAS, the EMYCIN knowledge acquisition module, asked the expert for new rules.

New systems are developped in more and more sophisticated environments such as KEE, Knowledge Kraft or Art, allowing object oriented or even hybrid representations (combining frames, objects and rules). These representations are much more expressive and readable. But their complexity suppose a good training for the user, and the knowledge engineer intervention. Much of the commercialized development environments are sold under "the acquisition tool" denomination and belong to that category.

3.1.2 Visual representation

An increasing number of tools proposes facilities to display graphic representations of the structured knowledge. Some tools focus even much more on the editing functionalities than on the semantical ones. But if the semantics is poor, as pretty and practical as the edition may be, the tool will provide no real help to carry out the acquisition. For instance, the KREME environment offers a lot of displaying devices, but it does not really guide the knowledge conceptualization and structuration [ABRETT,86].

3.1.3 Mediating and intermediary representations

Many existing knowledge acquistion systems integrate several elicitation techniques which require a specific knowledge representation (cf KRITON [LINSTER,88]). Generally, these tools are designed for the expert who gives his knowledge within a frame derived from an implicit model. The computational format is completely transparent for him.

Examples of such tools are ETS [BOOSE,86] or even AQUINAS,that expect the expert to describe the domain with objects and constructs from which inference rules are built up. It relies on an elicitation technique based on the Kelly's personnal constructs theory. This domain description is not always possible. But ETS proved to be efficient and practical for diagnosis or other synthesis problems. The knowledge representation offers here a temporary means to facilitate the expert communication with the software, and the knowledge analysis.

On the contrary, tools as KADS or MOLE permit knowledge elicitation and representation in a more definitive format. At the implementation level, the representation is transparent to the expert. He uses the items of an explicit domain model to enter his knowledge. The difficulty lies in the domain model definition. In the case of MORE, the model has been easily designed because the expertise dealt with medical diagnosis, which is quite easy to model.

Beside those specific representations, N. JOHNSON defines "mediating representations" as a means to prepare an explicit description of the expertise in a definitive form. It mediates between verbal data and standardised knowledge representation in expert systems. So they are both a medium of communication and a conceptual language to describe the task. They are supposed to be computer language independant.

One of these formalisms is the Systemic Grammar Network that supports the process of abstracting and conceptualising the expertise, and validating it later [JOHNSON,88].

Mediating representations are interesting supports to build a machine independant representation of the knowledge. These features meet most of the requirements that are specified for the MACAO representation in the next paragraph. For those reasons, the formalisms

in MACAO will be presented as a mediating formalism.

3.2 The knowledge representation specifications in MACAO

The fulfilment of the method properties puts basic constraints on the knowledge representation. They imply it to satisfy the following properties:

1. The representation must not compel the expert to adapt his expression to a rigid frame. It subsumes that the entire meaning given by the expert to his speech and behaviour can be kept thanks to an *important semantic capacity*. It also supposes that the representation is *close enough to natural language*, facilitates the communication with the expert, and has some compatibility with a presentation *easy to read* and understand (such as graphs).

2. The selected formalism must be able to represent any kind of knowledge, declarative as well as procedural and whatever the domain or the kind of problem may be.
   This choice is justified by our initial purpose of improving knowledge acquisition for multi-expert systems, which involves several domains and different kind of problems.

3. Furthermore, as no expert system environment has been set *a priori*, we thought the analysis relevance as well as the tool flexibility would be improved if it was not limited to a specific representation formalism. So we want the formalism in MACAO to be independant from the knowledge representation and process in the target expert system.

4. We also expect the representation to *play an active part in the knowledge acquisition process*. To this end, we want it to be both a practical and powerful support to the knowledge engineer analysis and to integrate some of the model concepts such as functional unit, determinant, procedure, ...

5. *Operationality and computability*: In order to test and validate not only the knowledge structuration, but also its dynamic properties, it must be possible to process the knowledge (by running inheritance or inference mechanisms for instance).

Form those specifications, we have derived general principals to define the knowledge representation in MACAO.

First, the criteria of flexibility and domain independance led us not to use a pre-defined structure, not even relying on an explicit domain or problem model. However, such a model could later be described in the terms of the representation semantics, and refered to.

To acheive the environment independance, we choosed to use an intermediary representation. But as we wanted it to simultaneously follow the method prerequisits, a *ad hoc* **"mediating representation"** seemed to suit better. Still, it is not completly computer language independant because it has been implemented.

The operationality choice raises the question whether the formalism has to include the mechanism that processes the knowledge or not. Generally, the more computable the knowledge representation is, the more competence is required to handle it, and the less appropriate the semantic capacity is. So a balance has to be found between calculability and readability.

As those criteria present much incompatibility, we felt the need to distinguish an internal (and operational) representation from an external (and readable) one.

Previously, we have assumed that the power of a tool includes the representation ability to take into account a part of the acquisition, mainly the structuration. In that case, it amounts to consider the capacity to connect knowledge structuration and display, internal and external representations. The use of a mediating representation subsumes that the knowledge can be easily translated into the target system formalism.

Now, the representation is specified enough for us to present its internal and external implementation.

4 KNOWLEDGE REPRESENTATION AND EDITION IN MACAO

It has often been suggested that analysis of an expertise at the "knowledge-level" can substantually improve the development of expert systems. Analysis at this level is not influenced by the notations used to encode the knowledge in the system. It respects the expert behaviour and the knowledge structure better than symbol level analysis. The first description of the knowledge level has been introduced by Newell in 1982, and has evolued since then into the definition of problem solving or domain models.

Knowledge analysis in MACAO is carried out at that level. As we wanted the acquisition tool to run automatically parts of it, we felt the need for a computer based representation scheme. It had to be the less rigid as possible, very close to the expert cognitive model and have a loose semantics. This main idea guided us in defining the internal representation described here, which is close to semantic networks.

4.1 Internal representation

As we said earlier, the internal representation has been defined in order to consider only the model and communication constraints without being influenced by a pre-defined data structure semantics. We gave priority to the representation readability, its closeness to natural language and its compatibility with a graphic edition, against complexity or operationality. As a consequence, we did not focus on the inference or inheritance mechanisms performances but rather on the achievement of the representation specifications.

To support the MACAO methodology, the Le_Lisp language has seemed to be a judicious choice for its capacity for symbolic processing, indispensable in MACAO, its flexible use and the possibility of its being extended. So, all the MACAO structures have been built from the Le_Lisp structured objects.

Moreover, we choosed the AIDA interface generator for its flexible use, the versability of its components and the quality of the interfaces it permits to develop.

After a short account of AIDA and the Le_Lisp objects, this paragraphe will present the structuration of data in MACAO and their utilization at each method stage.

4.1.1 The basic environment

The AIDA applications developpement environment enables the construction of graphic interfaces for programs written in Le_Lisp [Aida,88]. AIDA is built around an image description language that helps to the interface programming by offering elementary compo-

nents (scrollbar, menus ...). Those components react to different events specifed by the developper with the keybord or the mouse.

The AIDA image-oriented language allows to build elementary images (strings of characters, rectangles, icons ...) and combined images by setting them on a view in column, row ...

Some applications, built with AIDA, are provided with the system such as structured object editor, tree-editor, texteditor ... They can be used just as they are to construct complex applications, or improved for specifics needs.

With AIDA, it has been possible to design a friendly interface, well adapted to MACAO. The dialog level suits the expert or the knowledge engineer according to the stage. Effective viewings are proposed to the user at each stage of the method. He may decide at any time to modify the windows configuration on the screen, by moving them, changing their size or iconifying them.

4.1.2 The knowledge structures

They are defined corresponding to the model. The structures encoding are transparent to the expert who can see only their external presentation. On the contrary, the knowledge engineer may use the internal representation to modify or enrich the internal structure of data with specific functions.

- The Le_Lisp objects
  As we want to give priority to the semantic ability, the formalism chosen focuses on objects. Le_Lisp offers structured objects and permits to create a hierarchy of structure types with the help of substructures. Structured objects are made up of several untyped named fields. So, a substructure consists of two kinds of fields:
  -those inherited from the structure,
  -those which are characteristic to the substructure, declared during its definition.
  Specific functions, called methods, can be attached to each type of object. Those functions check their parameters type and are gathered in the type package.

- The structures in MACAO.
  They symbolize the knowledge items manipulated by the expert.
  The functional unit structure consists of:

  a name which identifies the functional unit,

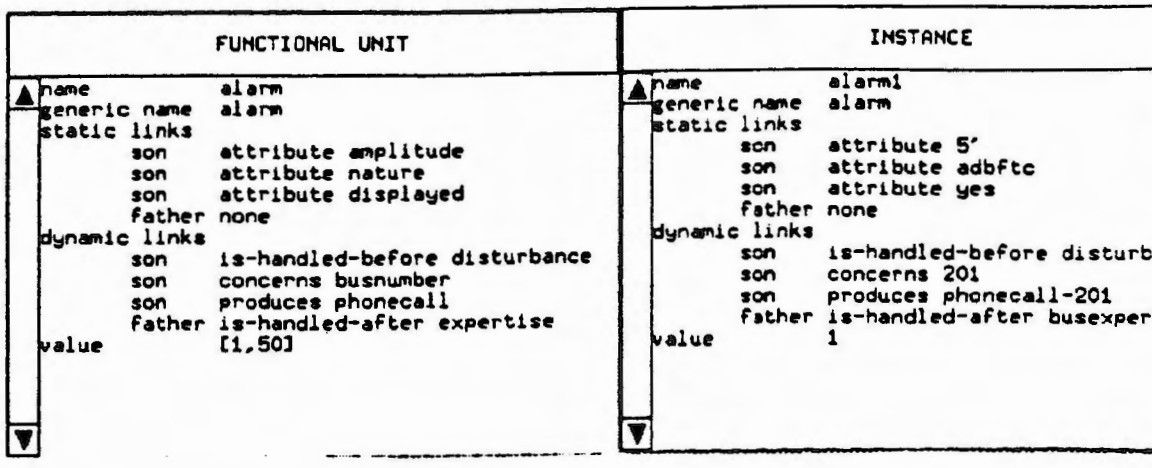
  a generic name which is the same for every functional unit instance,

  4 lists of links (son/father - static/dynamic) which represent the different kinds of connection with some other functional units.

  The value which contains either the possible values (for a generic functional unit), or the value itself (for an instance).

  The functional units are clustered in three classes, according to their conceptual level:
  -the generic functional units, with identical name and generic name,
  -the object functional units, representing a non generic domain object,
  -the functional unit instances, representing objects or concepts of the real world.

Here is an exemple of a functional unit and one of its instances:



Other structures, necessary to the knowledge representation have been defined:

the determinant is a subclass of functional unit. It represents an object or a concept which leads to different inferences depending on its value or the value of other functional units. In this way, it switches dynamically on the choice of the procedure to be run.

the link connects two functional units. It has a name and a role which can be either an explanation text or a Le_Lisp function. Links are clustered in two classes, according to their meaning:
- static links describe the world of the expertise: they connect an object to its different attributes, characteristics, properties or values. They express relations and structures located at the domain level as long as those relations are independant from the reasoning itself. Objects and concepts bound by such links form a kind of axiomatic structure that remains valid for the problems from the various categories;
- dynamic links describe the resolution process followed by the expert (a given object is calculated from, depends on, ... the ones it is linked to). They may represent the inferences processed when the expert uses a solving procedure. They are expected to differ from one category of problem to another.

the problem consists of a set of information bound to a problem solved by an expert. It has a declaration and belongs to a category. It is associated a glossary and a graph of functional unit instances.

the category of problem concerns a particular kind of problem executed bu the expert. It has a declaration and a list of procedures. It is associated its own conceptual graph.

the procedure is a logical sequence of functional units common to several problems of the category.

4.1.3 Possible functions on the structures

Those functions are accessible by the specific menu of each structure.
Functions which update lists of words, like glossaries or lexicons, are working like the ones of a text-editor. For each structure, it is possible to create, modify, instanciate or delete by using internal functions, written in Le_Lisp. These functions are handled transparently by the MACAO interface. Functional units can be linked, either by creating the link whith the help of the associated structure editor, or by drawing it on the graph.

4.2 External representation

4.2.1 Graphs definition

We believe that a graphic representation of the knowledge structure is a powerful device for the expert to control the formalized knowledge at each step. This representation facilitates the adaptation to the user's dialog and explanation levels in the man-machine interface of the following knowledge-based system. Furthermore, it provides a convenient tool for the expert's direct control of the knowledge consistency. To answer to these requirements, we choosed to represent the knowledge by several levels of oriented graphs, where the vertices are "functional units" and where the edges are dynamic links or static links depending on the kind of graph they belong. A vertex is make up by two kinds of fields:
-those which belong to the object or the concept it represents
-those which are specific to a vertex of a graph such as the displayed name and the connections with the other vertices.

The choice of the layout algorithm have been made according to the following criteria:

- it has the capability to take into account contraints on the layout such as: "a set of vertices must be placed on the external boundary of the drawing".
- it has a decent run-time which is critical in an interactrive application.
- the drawing occupies a reasonably small area [TAMASSIA,88].

4.2.2 Graphs conceptual levels and semantics

The knowledge is represented by two kinds of diagrams: graphs on the expertise level, where the vertice are the objects connected by links of different kinds, and graphs on the conceptual level, where the vertice are concepts. The graphs of the lower level can be considered as instantiations of the conceptual ones. This hierarchy of diagrams turns the final knowledge representation into a superposition of graphs. To homogenize this structuration, the vertice will be called "functional units" in both graphs. These units are the actual items handled by the expert and the knowledge engineer.

4.2.3 Possible functions on graphs

The user can make any desired alteration on graphs, like adding a link between two functional units, destroying one, or adding a functional unit. The up-dating appears on the graph with a particuliar drawing (bold font). These alterations will be effective on the internal representation only when the user will ask for their validation. At this moment, the up-dating commands will have a repercussion on the concerned internal structures, and

will provoque a real modification of the external representation. The positions of the vertice are calculated and the graphs are displayed again.

To do these modifications, the user dispose of friendly tools with which he can deplace a vertex or a graph in his totality, place a specific vertex in the middle of the image, link two functional units ...

The mechanism of knowledge treatment consists in going through the graph according to the dynamic links by applying the methods associated to the determinants.

4.3 The representation support to the method main stages:

In order to show how the representation supports the method, we present here the software paper and some of the functionalities allowed on the representation at each stage. To make the software running more explicit, we are going to refer to an example derived from our experience with bus-network controllers. In short, the paper of an operator is to oversee eight bus-lines the status of which can be displayed on a central computer screen. As the computer records data about the evolution of the network situation, the operator is given information such as the position of each bus on the line, bus deviation relating to the scheduled time...

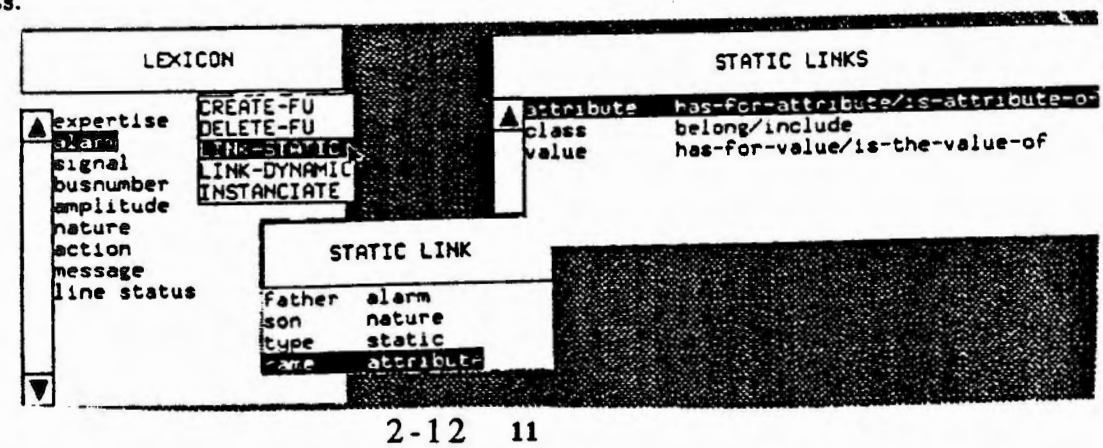
4.3.1 A guide to conceptualize and formalize

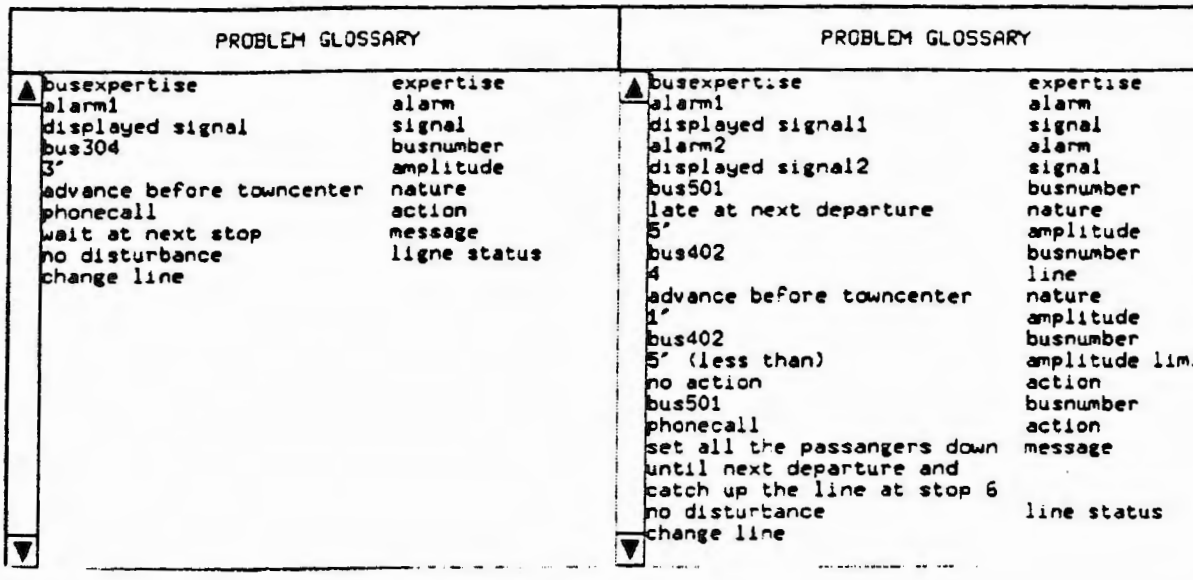
The following two views are examples of the text-editor windows that help the expert to enter the name of the objects and concepts that he handles when solving a problem. These lists correspond to the solving of similar problems, belonging to the same category: "the computer warns the operator by displaying alarms on the screen".

The left column of words, called the problem glossary, contains the objects names in the corresponding problem. The expert writes them all along the solving, but the knowledge engineer may add some missing word after he has listened to the recorded explanations. As a matter of fact, this up-dating is the major part of the conceptualization.

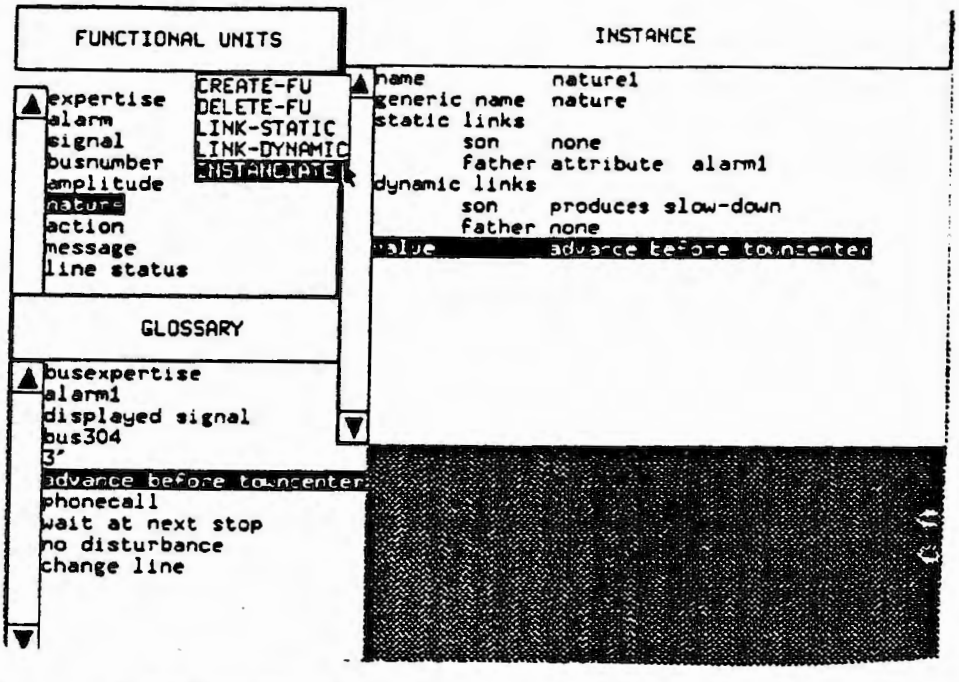
The right column gathers the generic names, picked in the domain lexicon, that are associated to the glossary words. They point out the concepts names that describe the problem solving at a more abstract level. The knowledge engineer formalizes objects and concepts later. To this end, he creates a functional unit to describe each word, generic for the lexicon words, instanciated for the glossary ones. He may modify or create functional units, connect or instanciate them as well.

For example, here are the views displayed when the knowledge engineer, (or the expert during the validation stage) wants to formalize that "One of the caracteristics of an alarm is its nature". We assume that "alarme" and "nature" are the name of existing functional units.





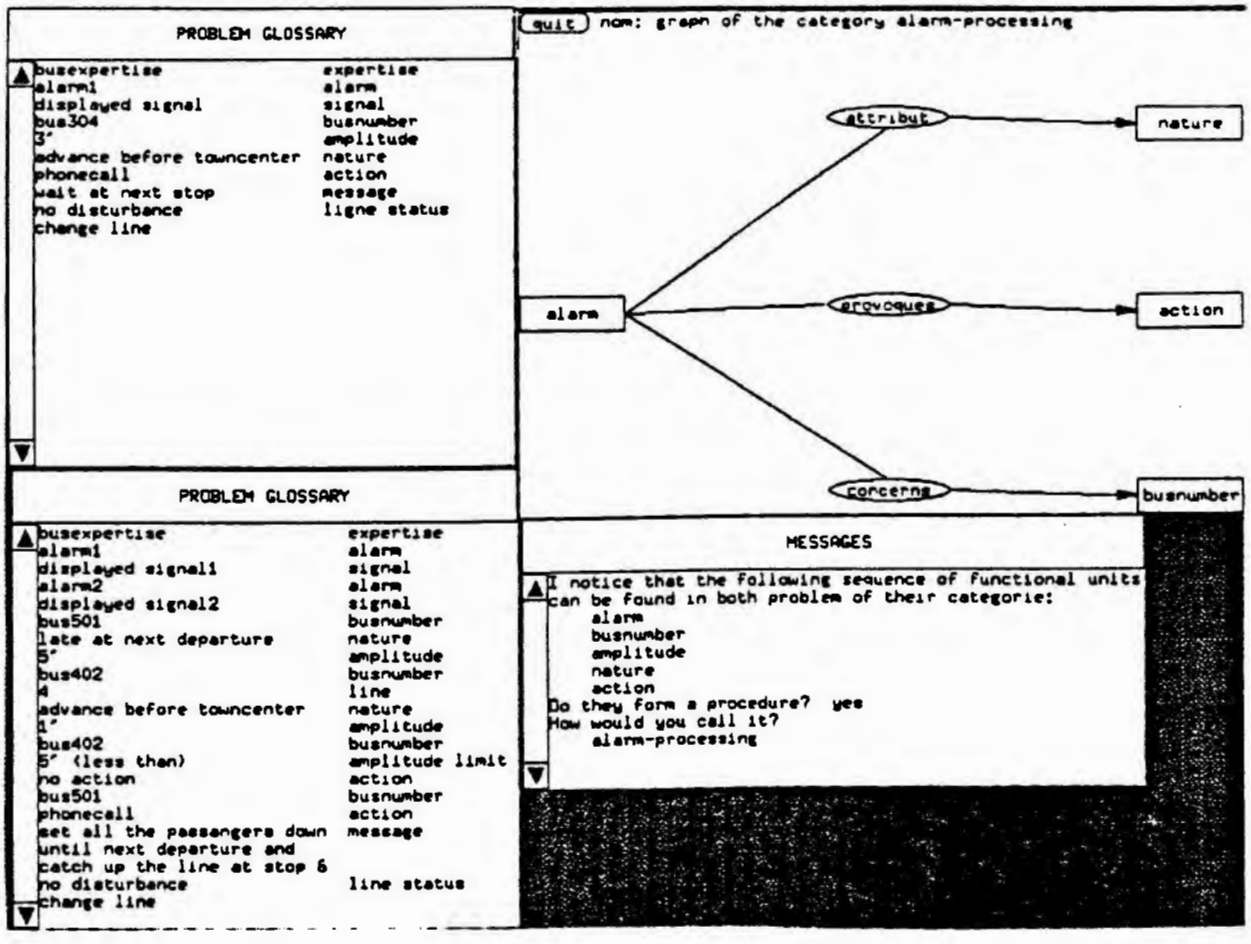
This second example shows how to create the functional unit corresponding to the nature of the first alarm displayed during a problem solving.



4.3.2 A support to the analysis

The knowledge analysis takes place in three times. The knowledge is conceptualized and structured first for each problem, then within each category and at last for the whole expertise domain. The use of the structures facilitates systematic comparision and rapid knowledge structuring.

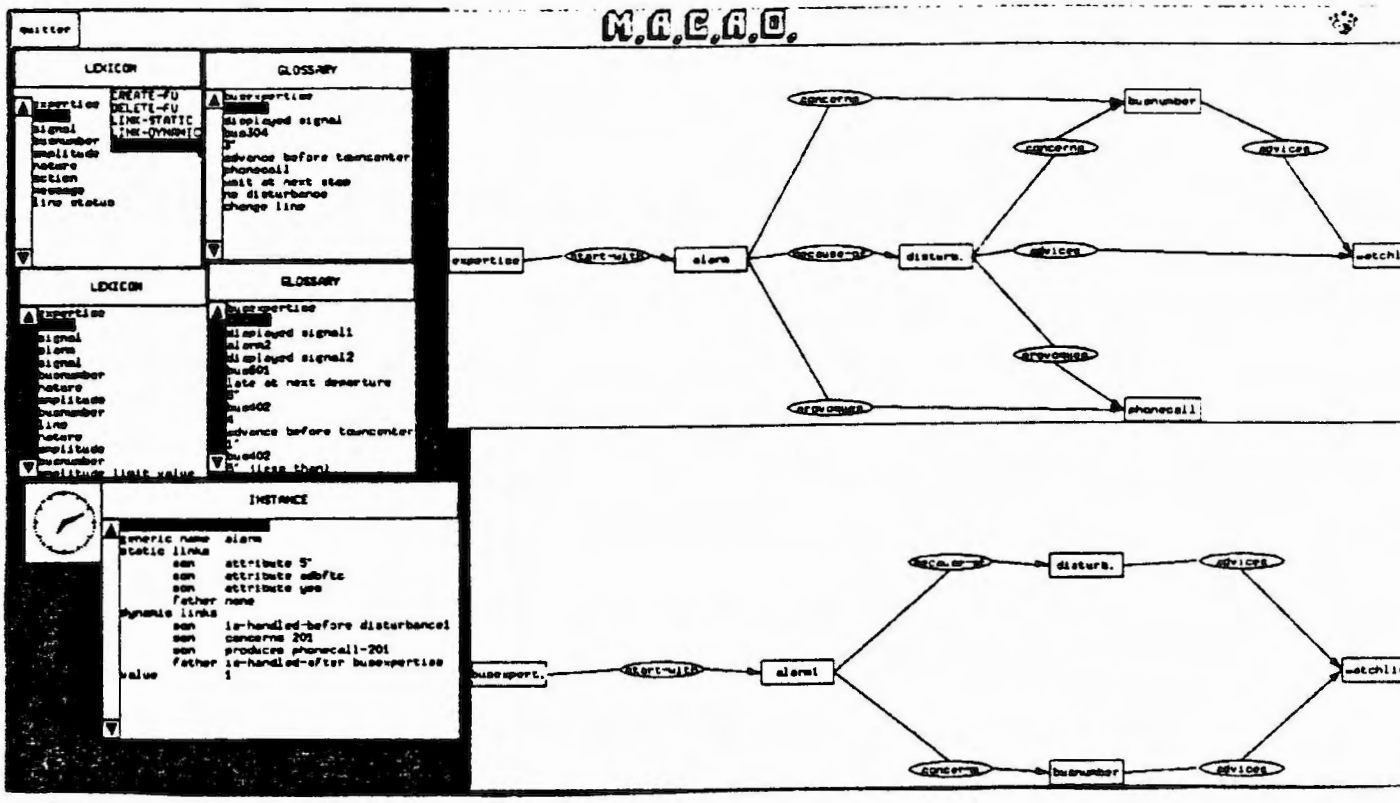
This example indicates how the software may suggest the knowledge engineer to identify a procedure: The problem glossaries comparision leads the system to display a message that warns the knowledge engineer, and the sequence of functional units it has found. The engineer can validate and label the procedure, or ignore the message, according to whether he agrees or not.



4.3.3 A means to ease validation

Beside the tool-box available in the conceptualization stage, the expert is proposed a set of functions in order to modify and validate the knowledge graphs. Those functions start up the graph updating process described before. The expert can go through the graph hierarchy or stay at a given level.

Here is an example of a graph updating, and its concequence on the assiociated structures.



5 DISCUSSION AND CONCLUSION

If we consider our motivations to design an acquisition methodology, our experience of the MACAO method definition and tool construction gave us the opportunity to answer to some of the field major questions.

Our collaboration with psychologists, even if sometimes difficult, helped us to define the method stages and the knowledge engineer's paper.

It may be possible for us to evaluate the interest and limit of such a model.

The associated tool is presently being experimented and validated to developp an application at MATRA-ESPACE TOULOUSE.

We have shown, in this paper, that the knowledge representation meets most of the requirements due to the method. In particular, it proposes an answer to both the development environment independance and to the delicate balance between readability and processing. We opted for a mediating representation based on internal structure and external graphs. This double representation enables the support of knowledge analysis. Most of all, the interface is accessible to the expert. We plan to improve the inference and inheritance mechanisms on the structures, and their display on graphs.

The graph theory seems to be a promising device to check knowledge consistency and completeness. Another possible development for MACAO will be the specifications of how to define a domain or problem model with an internal representation in order to improve the tool capicity for each given expertise.

References

[Aida,88] Manuel d'utilisation d'Aida. *Ilog* , 1988.

[ABRETT,86] ABRETT G., BURSTEIN M., The KREME Knowledge Editing Environment, AKAW, Banff Canada, Nov 1986.

[AUSSENAC,88] AUSSENAC N., MICHEZ B., MACAO: Application d'un modèle psychologique à la réalisation d'un outil d'aide à l'acquisition des connaissances. *Actes du colloque du CREIS*, Saint-Etienne, mai 1988.

[AUSSENAC,88b] AUSSENAC N., SOUBIE J. L., FRONTIN J., A knowledge acquisition tool for expertise transfer, *Proceedings of EKAW'88, GMD Studien Nr 143*, Bonn (RFA), 8.1-8.12 June 1988.

[BENCHIMOL,86] BENCHIMOL G., POMEROL J. C., LEVINE P., Systèmes experts dans l'entreprise *Ed. Hermès, "Gestion et Productique"*, Paris, 1986.

[BERRY,87] BERRY D., The problem of implice knowledge. *Expert Systems (4)*, 3, 114-151. August 1987.

[BONNET,86] BONNET A., HATON J. P., TRUONG NOC J. M. Systèmes experts : vers la maîtrise technique. *Interéditions*, 288 pages. 1986.

[BOOSE,86] BOOSE86 J. H., Expertise Transfer for Expert System Design. *Elsevier Ed., "Advances in Human Factors/Ergonomics"*, 3, 312 pages. 1986.

[BOOSE,87] BOOSE87 J. H., BRADSHAW J. M., Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. *Int. Journal of Man Machine Studies (26)*, 3-28. 1987.

[BOY,88] BOY G., FALLER B., SALLANTIN J., Acquisition et ratification des connaissances. *Actes des 2emes journées Internationales du PRC-GRECO, Intelligence Artificielle*, Toulouse, 321-356, mars 1988.

[BRACHMAN,79] BRACHMAN R. J., On the epistemological status of semantic networks. In N.V. Findler, editor, *Associatvie Networks*, Academic Press, New York, 1979.

[GAINES,88] GAINES B., Second generation knowledge acquiqition systems *Proceedings of EKAW'88, GMD Studien nr 143*, Bonn, p. 17.1-17.14 . June 1988.

[HAYWARD,87] HAYWARD S. A., WIELINGA B. J., BREUKER J., A., Structured Analysis of Knowledge. *Int. Journal of Man-Machine Studies (26)*, 487-498, 1987.

[HOC,88] HOC J. M., Psychologie Cognitive de la Planification, *Ed. PUF* , Paris, 1988. 197 pages.

[JOHNSON,88] JOHNSON N., TOMLINSON C.M., Knowledge elicitation for second generation expert systems *Proceedings of EKAW'88, GMD Studien Nr 143*, Bonn (RFA),23.1-23.10 ,June 1988.

[KAHN,85] KAHN G., NOWLAN S., MAC DERMOTT J., MORE: an intelligent knowledge acquisition tool. *Proceedings of Nineth International Conference on Artificial Intelligence*, Los Angeles, California, 637-639. 1985.

[LINSTER,88] LINSTER M, Kriton: A knowledge elicitation tool for expert systems, *Proceedings of EKAW'88, GMD Studien Nr 143*, Bonn (RFA),4.1-4.9 ,June 1988

[MICHEZ,86] MICHEZ B., Approche du fonctionnement cognitif: schéma empirique et schéma conceptuel dans le processus de structuration des connaissances. *Mémoire de DEA Université de Toulouse le Mirail.* 1986.

[MUSEN,86] MUSEN M., FAGAN L., COMBS D., SHORTLIFFE E. Using a domain model to drive an interactive knowledge editing tool. *Proceedings of AKAW'86*, Banff (CAN), Nov. 1986. p 33.0-33.11

[MUSEN,88] MUSEN M., Conceptual models of interactive knowledge acquisition tools, *of EKAW'88, GMD Studien Nr 143*, Bonn (RFA), June 1988. p 3.1 - 3.15

[REITMAN,87] REITMAN OLSON J., REUTER H., Extracting expertise from experts: Methods for knowledge acquisition. *Expert Systems (4)*, n 3, 152-167, August 1987.

[TAMASSIA,88] TAMASSIA R., DI BATTISTA G., BATINI C., Automatic graph drawing and readability of diagrams, *IEEE Transaction on systems, man and cybernetics*, Vol 18, N 1, 61-79, 1988.

[TRICKEY,88] TRICKEY H., Drag: a graph drawing system *Int. Conf. on Electronic Publishing Document Manipulation and Typography. Ed. Van Vliet*, Nice, 20-22 april 1988.

[VOGEL,88] VOGEL , Le génie cognitif. *Ed. Masson* , Paris, 1988.

[WIELINGA,84] WIELINGA B., J., BREUKER J., A., Interpretation of verbal data for knowledge acquisition, *ECAI 84, Advances in Artificial Intelligence*, T. O'Shea Ed., 41-50, 1984.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

ACQUIRING AND VERIFYING CONTROL KNOWLEDGE FOR A BLACKBOARD SYSTEM

Lawrence S. Baum, David B. Shema, John H. Boose, Jeffrey M. Bradshaw
Knowledge Systems Laboratory, 7L-64
Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, Seattle, WA 98124 USA

ABSTRACT

Knowledge acquisition technology can be tied to methods for verification of knowledge-based systems. Preliminary experiments have shown that testing and review of knowledge while it is being elicited can help overcome portions of this problem. This paper focuses on a knowledge-acquisition-oriented testing method for verifying knowledge-based systems. The method uses test cases supplied by an expert that include the expected or ideal behavior of the system given a set of conditions. Test cases are matched against actual system results to measure system performance in terms of accuracy. In certain instances, test cases are also used to automatically improve the knowledge base.

To extend this work to a new problem domain, we have linked Aquinas, a knowledge acquisition workbench, to Erasmus, a blackboard system, for an aircraft cockpit information management application. This is a prototype of a future real-time system. Aquinas interviews domain experts and generates knowledge sources for Erasmus that contain blackboard control heuristic information. This control information is used by the scheduler in Erasmus to select appropriate knowledge sources. Domain experts build test cases in Aquinas; these are automatically matched against Erasmus reports to measure control knowledge performance against the experts' expectations of performance. Aquinas also automatically tries to improve the experts' original knowledge bases by changing certain information and seeing if it improves knowledge base performance.

We hope to show that the coupling of Aquinas and Erasmus will provide (1) an environment for acquiring and verifying control knowledge for a real-time application (traditionally such knowledge has been difficult to acquire and verify), (2) a testing and performance measurement method that addresses aspects of verification problems, and (3) a method for automatically improving control structures based on test cases. Preliminary results of this investigation are reported here; Aquinas seems to be an effective aid in eliciting and verifying control knowledge structures for Erasmus.

1.0 INTRODUCTION

This section discusses the problem of verifying knowledge-based systems, the application and its underlying technology base, and a tool used to acquire domain control knowledge from the expert.

In this work we are attempting to verify control knowledge elicited from a domain expert for an application that uses the blackboard paradigm. Traditional rule-based verification techniques cannot be used for blackboard applications, since these applications utilize knowledge representation schemes beyond traditional IF-THEN rules, such as frames and procedures.

There is little agreement on the nature of knowledge base verification (Nazareth, 1989). Some see verification as a problem of proving that knowledge bases are consistent, complete, and correct. This work typically focuses on methods for checking rule consistency, rule circularity and some types of rule completeness (Suwa, Scott, and Shortliffe, 1982; Marek, 1986; Mars and Miller, 1986; Cragun and Steudel, 1987; Nguyen, 1987). A rule consistency check highlights contradictions between rules. A rule circularity check shows circular paths that a backward or forward chaining inference engine might follow. Completeness checkers usually try to build models of typical rules and determine if clauses are missing.

Others see verification as the problem of proving that a system meets a given set of specifications (Green and Keyes, 1987; Geissman and Schultz, 1988). Both problems are important to the development of knowledge-based systems.

While methods that check internal correctness can help find minor problems and inconsistencies in a knowledge base, they do not address overall questions about whether the knowledge base is solving the specified task. These problems address issues of *necessity* and *sufficiency*:

a) Are the knowledge base concepts (such as rule variables or frame slots) that are present *necessary*?

b) Is the information in the knowledge base *sufficient* to solve a given set of problems?

c) What is the range of problems for which the knowledge base performs well in a given domain?

d) Is some knowledge more important?

e) Should some knowledge be broken down further for more precise control?

f) Given target behavior, can the system help improve itself?

Experts, knowledge engineers, project managers and others associated with knowledge-based system projects lack techniques to accurately answer such critical questions.

1.1 Cockpit Information Manger

Because of the increasing complexities of modern weapon systems and missions, aircrews are faced with demands that often exceed their ability to perform as they have in the past. Advanced avionics systems are capable of gathering and processing much greater volumes of data, much of which is essential to mission success. Consequently, crews must interpret a

larger amount of data, must deal with increasingly complex interactions with onboard systems, and have considerably less time in which to perform.

A cockpit information manager is an interface between the pilot and the aircraft avionics systems. It must assimilate information from a number of subsystems, including mission planning, external situation assessment, and internal situation assessment (diagnostics). It must decide, for a given situation, what information is most important for the pilot to see, how that information should be displayed, and how much detail to show. In addition, the cockpit information manager must respond rapidly to instructions from the pilot. The cockpit information manager is also responsible for determining where the information should be displayed.

For example, if the pilot requests information on aircraft in his immediate vicinity, the cockpit information manager must decide which objects to display, the symbology to use for those objects, and the amount of detail to provide for each object. Too much detail can result in a display that is too cluttered for the pilot to assimilate the information displayed. Too little detail can mean a display with insufficient information for the pilot. Furthermore, the cockpit information manager must make its decisions in real time. On the other hand, the cockpit information manager is not responsible for recognizing or reacting to threats; those tasks are the responsibility of the situation assessor and the mission planner. The task of the cockpit information manager is to notify the pilot in the most appropriate way to allow him to react with minimal effort and distraction.

1.2 Blackboard Technology

Blackboard architectures involve three fundamental components (Nii, 1986):

a) The *blackboard* is an explicit, shared repository for problem-solving data.

b) The *knowledge sources* are independent agents which work together to solve the given problem.

c) The *control mechanism*, or scheduler, has the responsibility of deciding which of the activated knowledge sources should be allowed to act next. This decision will normally depend not only on the choices (knowledge source activations) available to the scheduler, but also on the current state of the problem solving.

1.3 The Erasmus Blackboard System

Erasmus (Baum et al., 1989) is a framework for constructing applications using the blackboard model. Erasmus executes in the following cycle:

a) The *scheduler* invokes one or more knowledge source activations (KSA) from its agenda. Erasmus uses a hierarchy of objects on the control blackboard to prioritize KSAs. This type of information is the subject of our experiments.

b) The chosen KSAs are executed, either sequentially or in parallel, depending on the implementation. What a knowledge source activation does is entirely application dependent; to the blackboard framework, knowledge sources are "black boxes."

Usually, a KSA will read data from the blackboard, process that data in some fashion (perhaps affecting the external world) and write results of some sort back to the blackboard. These modifications are called *events*.

c) For each such event, the blackboard manager determines which of the application's knowledge sources are interested in it. Knowledge sources have trigger conditions that describe the kinds of events to which they should respond. The triggering of knowledge sources results in additional KSAs for the scheduler's agenda.

As this sequence repeats, the "problem solution" evolves on the blackboard.

1.4 Control Knowledge in Erasmus

The separation of control knowledge from domain knowledge is central to the blackboard paradigm. System behavior depends, of course, on both types of knowledge. Nevertheless, without the appropriate control knowledge, the system will not perform correctly. This is the knowledge that we are interested in acquiring and verifying.

Erasmus uses a hierarchy to represent control knowledge for the scheduler (Figure 1):

a. A *heuristic* rates KSAs. The rating of a KSA might be determined simply as a function of what knowledge source it instantiates, or it might involve a complex computation based upon the current state of the blackboard and the local context of the KSA. Heuristics are implemented as Lisp functions in Erasmus.

b. A *focus* is implemented by a set of heuristics. The purpose of a focus is to direct problem solving by focusing attention on some goal or an area of the blackboard or other conceptual entity using domain-specific knowledge about what domain knowledge sources do and how well they do it. Each focus has a goal.

c. A *strategy* is implemented as a series of *prescriptions*, which in turn are sets of foci. At any point, one prescription is current for a strategy. The current prescription is active until each of its foci has achieved its goal. When all of the foci for a prescription have reached their goals, that prescription expires and the next one on the list becomes active. Control knowledge sources are free to modify the goal of a strategy or to change the currently operative strategy.

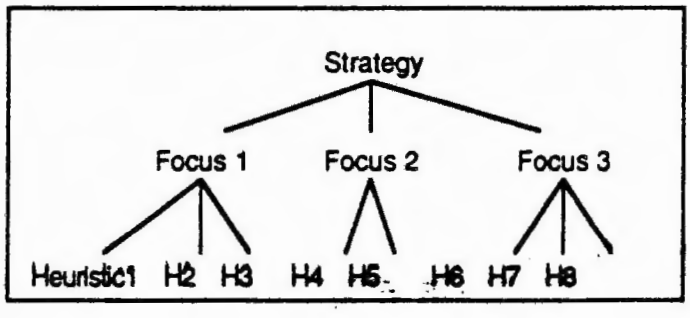


Figure 1. Erasmus represents control information hierarchically.

The priority of a KSA is computed as a weighted sum of the ratings of the KSA by each operative focus. Those focus ratings are in turn computed as a weighted sum of the ratings of the KSA by the heuristics that implement the focus. Weights of both foci and heuristics can be modified by control knowledge sources as conditions warrant.

The objects on the control blackboard are placed there and modified by *control* knowledge sources, which are triggered and invoked in exactly the same fashion as *domain* knowledge sources (Hayes-Roth, 1985).

1.5 Control Knowledge Problems

The design of the control objects (i.e., the heuristic Lisp functions and the various goal functions) and the control knowledge sources that manipulate them is a difficult task. In each cycle, the scheduler is faced with a set of KSAs and must choose among them. The control objects determine this selection. Though Erasmus currently provides a set of tools for monitoring this decision process and interactively experimenting with the weights assigned to heuristics and foci, it lacks a tool for acquiring, verifying, and maintaining the control knowledge.

In the current environment, knowledge engineers and experts (pilots) change control heuristics by editing Lisp functions. Previously constructed scenarios are then run on a simulator. The simulator consults the blackboard asynchronously to determine the configuration of a cockpit display. Knowledge engineers and experts observe the cockpit display and accompanying transcripts to determine if the overall system behavior is appropriate. They may also modify a scenario by entering pilot commands while the simulator is running.

The process of determining the correctness of the system's behavior is subjective. Often several different behaviors could be appropriate in response to a given situation. The Erasmus scheduler may execute hundreds of decision cycles during a typical scenario. Because of the entry of pilot commands and the asynchrony of system components, the results of running a scenario are generally unrepeatable.

As the heuristic Lisp functions grow in complexity, it is difficult for experts and knowledge engineers to —

(1) Make knowledge base improvements based on observed incorrect decisions by the blackboard scheduler, especially when such decisions are only seen after many cycles of the Erasmus blackboard system have been run.

(2) Measure blackboard control knowledge performance (accuracy) during testing.

We are attempting to address these specific problems by using a knowledge acquisition workbench, Aquinas, to represent control information and provide a more rigorous testing environment.

1.6 Aquinas: A Knowledge Acquisition Workbench

Aquinas, an expanded version of the Expertise Transfer System (ETS; Boose, 1984, 1985, 1986a), combines ideas from psychology and knowledge-based systems research to support knowledge acquisition (Figure 2). 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, use of constraints during inference, and providing process guidance (Boose and Bradshaw, 1987; Boose, Bradshaw, and Shema, 1989). Aquinas interviews experts and helps them analyze, test, and refine knowledge. 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.

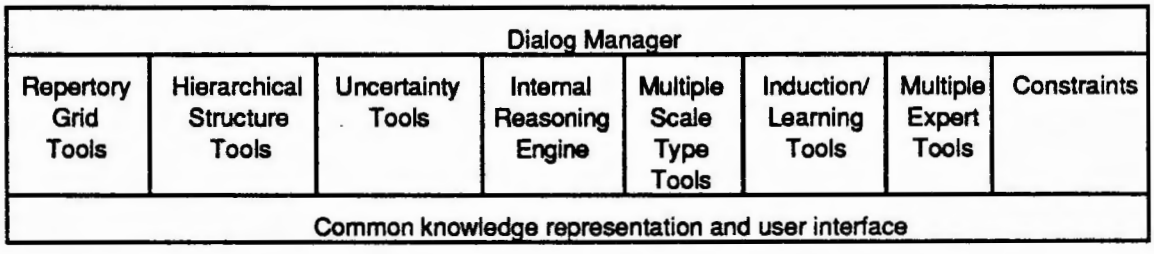


Figure 2. Aquinas is a knowledge acquisition workbench containing a variety of elicitation, analysis, and testing tools.

Applicable refinement methods in Aquinas include implication and similarity analyses, completeness checking, hole filling, cluster analyses, generalization, automatic rule production, internal testing and debugging aids, and graphic representation transformation. 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. Recent progress on Aquinas has been in the areas of knowledge base performance measurement, knowledge base maintenance, interacting trait constraints, consultation graphics, and elicitation of 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 are also employed by Aquinas (Boose, Bradshaw, and Shema, 1989).

2.0 USING AQUINAS TO ACQUIRE ERASMUS CONTROL KNOWLEDGE

2.1 Project Goal and Phases

We are attempting to work with control knowledge for a knowledge-based system application. Traditionally such knowledge has been difficult to acquire and verify. The goals of this project are to —

- Make it easier and more efficient to acquire and verify Erasmus control knowledge.
- Improve blackboard application performance (accuracy, not efficiency) through the use of knowledge acquisition tools.

Our approach is to tie knowledge acquisition technology to methods for verification of knowledge-based systems. Preliminary experiments have shown that testing and review of knowledge while it is being elicited can help reduce verification problems (Shema and Boose 1988). In addition, use of intermediate knowledge representations in knowledge acquisition tools (e.g., repertory grids, network representations) makes knowledge base creation, refinement, and maintenance more efficient and effective.

The project is being conducted in three phases; the "baseline system" is the original Erasmus application without Aquinas:

Phase 1. Match the baseline system's behavior in a new system using Aquinas.

During this phase, Aquinas was used to represent Erasmus heuristic control knowledge in grids and automatically generate the same Lisp code that knowledge engineers generated for the baseline system. This proved that it is feasible to represent Erasmus control knowledge in Aquinas. Grids were augmented to handle *utilities*, a new type of knowledge necessary for efficiently representing the heuristics. This is demonstrated in Section 2.2.

Phase 2. Implement testing and analysis methods in the new system.

During this current phase, analysis and testing methods in Aquinas will be modified to handle augmented grids. We will also use test-case information from experts to measure system performance (accuracy) and use Aquinas to automatically adjust and improve heuristic weights and utilities. In this phase we hope to discover how large a test-case library is needed to reach the project goals. We hope to develop methods based on techniques for checking table completeness to automatically examine the test-case library and judge its degree of application coverage.

Phase 3. Improve the new system's behavior.

In the third phase, we will use analyses in Aquinas and simulator observations to measurably improve the new system's behavior. We will also attempt to use Aquinas to acquire knowledge for planned increases in the complexity of the situations that the application will be able to handle.

2.2 Using Aquinas To Acquire Control Knowledge

Figure 3 shows the overall strategy for linking Aquinas and Erasmus. In Phase 1, Aquinas was augmented to allow the expert to enter heuristic control information into a grid. Then Aquinas was modified to directly generate Lisp code that Erasmus could read. This Lisp code exactly matched the code in the baseline system so that we would be guaranteed to match the baseline system's performance.

In Phase 2, Aquinas will improve heuristic weights and utilities using a test-case library produced by the experts. This library will specify requirements for system behavior in terms of the accuracy of results. Aquinas will adjust weights and utilities to match the behavior prescribed by the test cases. After adjusting weights and utilities, Aquinas will produce Lisp code for Erasmus.

Using the new Lisp code, Erasmus works with the simulator, producing a record of key decisions. These records are read back into Aquinas and used to measure system performance against a set of test cases constructed by the expert. Experts and knowledge engineers observe the simulator's behavior and make inferences about problems that may exist. They then modify grid information in Aquinas, rather than rewriting Lisp code. Aquinas tools may be used to analyze the changes. Then, Aquinas produces new Lisp code, and the simulator is run again. The new system behavior may lead to new modifications in Aquinas, until the system reaches a prescribed level of performance.

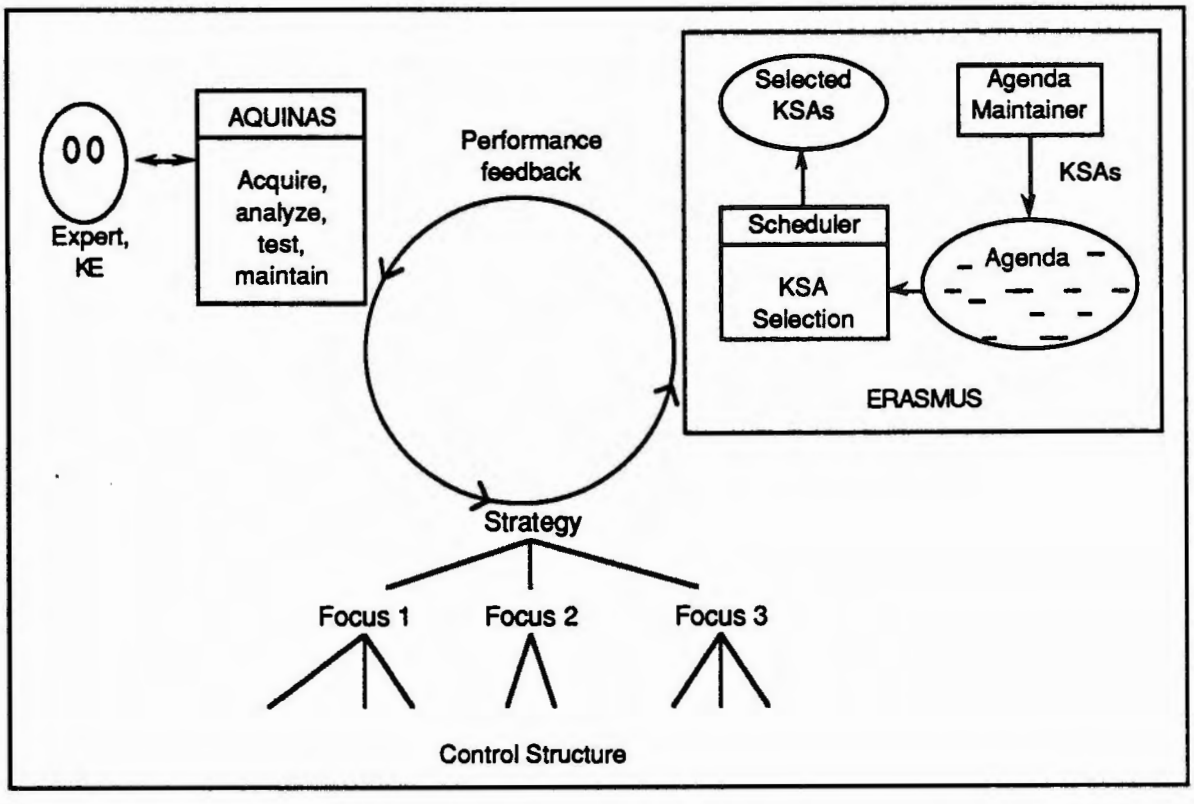


Figure 3. The Aquinas / Erasmus knowledge source generation and feedback cycle.

The potential advantages of this new system for verification and maintenance over the baseline system are:

a) Heuristic knowledge is maintained and modified in a format (grids) more comprehensible than Lisp code.

b) Analysis tools may be used with grids to find redundant information, make generalizations, identify gaps in knowledge, find the best and worst possible behavior of the information in a grid, and so on.

c) It is easier to build control heuristics from scratch using interviewing techniques based on Personal Construct Psychology (Kelly, 1955).

d) Changes in system performance due to modifications in control knowledge can be measured using test cases.

e) Aquinas may be used to automatically improve system behavior by adjusting weights and utilities associated with heuristics. In the baseline system, this is done by hand.

The expert used Aquinas to represent heuristic control knowledge. A grid that replicated the knowledge contained in the baseline system is shown in Figure 4. Aquinas helped the expert pilot analyze the potential best performance of the variables used in heuristics by seeing how well they could possibly discriminate between and choose knowledge sources.

					DBS SOLUTION TRAIT
TARGET	COMMAND	TARGET	OLD-TARGET	OTHER	1. (1.00) CHECK-COMMAND
HANDLE	HANDLE	HANDLE	OTHER	OTHER	2. (0.77) CHECK-FORMAT-CHANGE
OTHER	RANGE	RANGE	CENTER	CENTER	3. (1.00) CHECK-SCALE-CHANGE
TARGET	COMMAND	COMMAND	MFD-CONTROL	OTHER	4. (0.90) FORMAT-MAP
POP-UP	NEW	NEW	NEW	VALVE	5. (0.90) CHECK-FOR-NEW
OLD-SAM	OLD-SAM	OTHER	OTHER	OLD-SAM	6. (1.00) CHECK-FOR-OLD
OTHER	OTHER	OTHER	KILL-ASSESS	OTHER	8. (0.50) KILL-ASSESSMENT
OTHER	OTHER	OTHER	OTHER	OTHER	9. (0.50) INPUT-CLUSTER
|	|	|	|	|	
|	|	|	|	5. KSAR5	
|	|	|	4. KSAR4		
|	|	3. KSAR3			
|	2. KSAR2				
KSAR1					

Figure 4. Aquinas repertory grid for selecting knowledge sources.

We are currently updating Aquinas's analysis tools to handle augmented grids. When this is accomplished, we will be able to help the expert identify redundant and incomplete knowledge.

Figure 5 shows a portion of the Lisp code generated by Aquinas for Erasmus.

```
defun CHECK-COMMAND (ksar)
  (cond
    ((equal (send ksar :knowledge-source) 'COMMAND) 1.00)
    ((equal (send ksar :knowledge-source) 'TARGET) 1.00)
    ((equal (send ksar :knowledge-source) 'OLD-TARGET) 1.00)
    ((equal (send ksar :knowledge-source) 'OTHER) 0.00)
    (t 0)))
defun CHECK-FORMAT-CHANGE (ksar)
  (cond
    ((equal (send ksar :knowledge-source) 'HANDLES-CURRENT-FORMAT) 0.95)
    ((equal (send ksar :knowledge-source) 'OTHER) 0.00)
    (t 0)))
defun CHECK-SCALE-CHANGE (ksar)
  (cond
    ((equal (send ksar :knowledge-source) 'RANGE) 0.90)
    ((equal (send ksar :knowledge-source) 'CENTER-TRACK) 0.85)
    ((equal (send ksar :knowledge-source) 'OTHER) 0.00)
    (t 0)))
```

Figure 5. Control heuristics generated by Aquinas for Erasmus.

2.3 Verification: Testing and Performance Measurement

Aquinas measures knowledge base performance (accuracy) using test cases. For the Erasmus scheduler, a test case includes a snapshot of the blackboard environment and the expected ordering of the triggered knowledge sources. The expert supplies the ordering given the environment. Since it is not feasible to supply this information for every potential scheduler decision (there are usually hundreds of decision cycles in a single simulator scenario

and there are currently ten scenarios used for testing the baseline system), the expert must choose cycles that represent critical decision points. In practice this is not difficult. Information for each test case is stored in a file.

We are now in the process of building the test-case library. System performance for each test case will be measured by comparing the results of the Erasmus system's scheduling decisions with the expert's expectations (a correlation score is computed). Results for sets of test cases will be stored, and the impact of changes to the knowledge base will be measured by automatically rerunning the test cases.

Snapshots of the scheduler's decisions are analyzed by Aquinas in batch mode. Changes made to the knowledge base to improve the performance of one or a few test cases are automatically tested on all the previous cases. The expert will see whether new changes result in overall improvement by reviewing the performance measurement report.

2.4 Improving System Behavior

2.4.1 Automatically Improving a Grid

In Phase 2, we are currently collecting test-case information from experts. Test cases will be used by Aquinas to improve performance before Lisp code is generated. In Phase 1, Lisp code was generated without this enhancement, matching existing performance. We expect that using Aquinas to adjust weights will improve system performance in Phases 2 and 3.

2.4.2 Observable Problems and Corrections

In the baseline system, knowledge engineers and experts observe the cockpit display and accompanying transcripts to determine if the overall system behavior is appropriate. They make changes to Lisp code (control heuristics and knowledge sources) to test system improvement ideas.

In the new system knowledge engineers and experts will still observe the cockpit display and accompanying transcripts to determine the appropriateness of system behavior, but they will have a set of tools at their disposal to help them more efficiently test system improvement ideas. Additionally, Aquinas may be able to suggest changes based on static and dynamic analysis of the grids.

Observed performance problems (accuracy) may come from several areas. Some of these areas will be tractable to improvement using Aquinas:

a) Test cases. Some critical decision points might not be represented in the test-case library. A new test that covered this decision would be added. This process will be facilitated using the Erasmus scheduler's snapshot recording mechanism. The expert may also wish to modify existing test cases (knowledge source ordering or blackboard environment variables). Once new cases are added or old ones are modified, Aquinas will further improve the grid and generate Lisp code for subsequent testing by the simulator.

b) Heuristics. Some heuristics may be redundant or highly dependent. This could lead to flaws in reasoning if such heuristics are "doubly" weighted. Aquinas can help identify

such redundant heuristics (similarity analysis, cluster analysis). It is also possible that performance could be improved by adding heuristics. Aquinas can help identify these points through static analysis (similarity analysis, implication analysis, table completeness checking) and dynamic analysis (automatic grid improvement may lead to new terms). Aquinas also can suggest improvements to heuristic utilities and weights.

Other problems that are less tractable using these methods include non-repeatability of simulator situations due to asynchronous communication and dynamic pilot commands, and omission of or errors in the knowledge sources themselves.

3.0 DISCUSSION

We are investigating methods for verifying knowledge bases by using knowledge structuring by problem case, knowledge testing tools, and verification by automatic comparison of consultation results with expert expectations, sensitivity analysis, and automatic improvement of repertory grid knowledge bases. Preliminary experiments using these techniques for less complex problems have proved successful; we are attempting to apply them to more complex problems that utilize heuristic control knowledge in a cyclic environment.

So far we have duplicated the behavior of a baseline cyclic application by representing heuristic control information in Aquinas and generating Lisp code for a blackboard system. This is the first time we have used Aquinas to represent such information; we augmented Aquinas to represent heuristic control information in grids. We are collecting test cases to apply our knowledge acquisition-based testing and improvement techniques in the next phase.

These methods attempt to solve problems of *necessity* and *sufficiency* that arise when building knowledge-based systems. These problems appear to be fundamental aspects of verification problems.

Our work is addressing the following issues:

a) Are the knowledge base concepts that are present necessary?

Sensitivity analysis and static analysis techniques in Aquinas will be used to help identify redundant heuristic information in Erasmus that did not contribute to solving test-case problems.

b) Is the information in the knowledge base sufficient to solve a given set of problems?

Automatic knowledge base improvement in Aquinas will help identify new Erasmus control knowledge that will improve performance (problem-solving accuracy). In addition, Aquinas techniques associated with static testing (such as implication analysis and similarity analysis) can show where new information is needed.

c) What is the range of problems for which the knowledge base performs well in a given domain?

Test case performance measurements in Aquinas will show specifically where Erasmus control knowledge succeeds and fails on a problem-by-problem basis.

d) How can performance (problem-solving accuracy) be measured? When a knowledge base is changed, how does the expert, knowledge engineer, or maintainer know whether the system has been improved or damaged?

The test-case case mechanism in Aquinas will be used to measure positive and negative changes in Erasmus performance when modifications are made to the control heuristics.

e) Is some control knowledge more important? Should some knowledge be broken down further for more precise control?

Sensitivity analysis in Aquinas will help prioritize Erasmus control heuristics and identify those that should be broken down further when necessary for additional precision.

f) Given target behavior, can the system help improve itself?

Automatic knowledge base improvement methods in Aquinas will be applied to Erasmus test-case sets to automatically improve performance and suggest missing, important problem-solving information.

We hope to show that the coupling of Aquinas and Erasmus will provide (1) an environment for acquiring heuristic control knowledge for a blackboard application, (2) a testing and performance method that addresses aspects of verification problems, and (3) a method for automatically improving Erasmus heuristic control structures based on test cases to alleviate maintenance problems.

REFERENCES

Baum, L.S., Dodhiawala, R., Jagannathan, V. (1989). "The Erasmus System," *Blackboard Architectures and Applications*, V. Jagannathan, Raj Dodhiawala and Lawrence Baum, (eds.), Academic Press, 1989, pp. 347-370.

Boose, J. H. (1984). Personal Construct Theory and the Transfer of Human Expertise, *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, Austin, Texas, pp. 27-33.

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

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

Boose, J. H. (1986b). Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain. *Future Computing Systems,* Vol. 1, No. 2, p. 191-216.

Boose, J. H. (1988). Uses of Repertory Grid-Centred Knowledge Acquisition Tools for Knowledge-Based Systems. *International Journal of Man-Machine Studies,* Vol. 29, No. 3, p. 287-310.

Boose, J. H. and Bradshaw, J. M. (1988). Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems. *International Journal of Man-Machine Studies,* Vol. 26, No. 1, p. 3-28; also in Boose, J. H. and Gaines, B. R. (eds), Knowledge-Based Systems Vol.2: Knowledge Acquisition Tools for Expert Systems, London: Academic Press, 1988, p. 39-64.

Boose, J. H., Bradshaw, J. M. and Shema, D. B. (1989). Recent Progress in Aquinas: A Knowledge Acquisition Workbench. *Knowledge Acquisition: An International Journal,* in press.

Bradshaw, J. M. and Boose, J. H. (1989). Decision Analysis Techniques Combining for Knowledge Acquisition: Combining Information and Preferences Using AQUINAS. *International Journal of Man-Machine Studies,* in press.

Clancey, W. (1986). Heuristic Analysis. in J. Kowalik (ed). *Knowledge-Based Problem Solving.* New York: Prentice Hall.

Cragun, B. and Steudel, H. J. (1987). A Decision Table-Based Processor for Checking Consistency and Completeness of Rule-Based Expert Systems. *International Journal of Man-Machine Studies,* Vol. 26, No. 3, p. 333-348.

Geissman, J. R. and Schultz, R. D. (1988). Verification and Validation of Expert Systems. *AI Expert,* Vol. 3, No. 2, p. 26-33.

Green, C. J. and Keynes, M. M. (1987). Verification and Validation of Expert Systems. *Proceedings of the Western Conference on Expert Systems (WESTEX-87),* Anaheim, California, p. 38-43.

Hayes-Roth, B. (1988). A Blackboard Architecture for Control. *Artificial Intelligence,* Vol. 26, No. 3, p. 251-321, July.

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

Kitto, C.M. and Boose, J. H. (1988). Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs. in Boose, J. H. and Gaines, B. R. (eds), Knowledge-Based Systems Vol.2: Knowledge Acquisition Tools for Expert Systems, London: Academic Press.

Marek, W. (1986). Consistency and Completeness of Knowledge-Based Systems. *Proceedings of the First International Conference on Expert Database Systems,* Anaheim, Charleston, S.C., pp. 129-126.

Mars, N. J. and Miller, P. L. (1986). Tools for Knowledge Acquisition and Verification in Medicine. *Proceedings of the Tenth Annual Symposium on Computer Applications in Medical Care,* Washington D.C., October, pp. 36-42.

Nazareth, D. L. (1989). Issues in the Verification of Rule-Based Systems. *International Journal of Man-Machine Studies,* in press.

Nguyen, T. A. (1987). Verifying Consistency of Production Systems. *Proceedings of the Third IEEE Conference on Artificial Intelligence Applications,* Orlando, FL, February, pp. 4-8.

Nii, H. P. (1986). Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of the Blackboard Architecture. *AI Magazine,* Vol. 7, No. 2, p. 38-53.

Shaw, M. L. G. and Gaines, B. R. (1987). Techniques for Knowledge Acquisition and Transfer. *International Journal of Man-Machine Studies,* Vol. 27, No. 3, p. 251-280.

Shema, D. B. and Boose, J. H. (1988). Refinig Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism. *International Journal of Man-Machine Studies,* Vol. 29, No. 4, p. 447-460.

Suwa, Scott, A.C. and Shortliffe, E.H. (1982). An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System. *AI Magazine,* Summer, p. 16-21.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

LEARNING FLEXIBLE CONCEPTS THROUGH A SEARCH FOR SIMPLER BUT STILL ACCURATE DESCRIPTIONS

Francesco Bergadano* , Stan Matwin** , Ryszard S. Michalski+, Jianping Zhang+

*Dipartimento di Informatica, Universita di Torino, Corso Svizzera 185, Torino,Italy
**Department of Computer Science, University of Ottawa, Ottawa, Ont. K1N 6N5 Canada
+Artificial Intelligence Center, George Mason University, Fairfax, VA 22030, USA

ABSTRACT

This paper describes a method for learning flexible concepts, which are defined as concepts that lack precise definition and are context-dependent. The method is based on a *two-tiered concept representation.* In such a representation the first tier, called the *Base Concept Representation,* describes typical properties of a concept in an explicit, comprehensible, and efficient form. The second tier, called the *Inferential Concept Interpretation*, defines allowable transformations of the concept under different contexts.

In the method, the first tier is created in two stages. In the first stage, a complete and consistent description of the concept is learned by applying the inductive learning methodology (AQ and INDUCE) to examples of varying typicality. In the second stage, so obtained description is improved through a heuristic search, employing a *description quality criterion.* During the search, the concept description is simplified by applying truncation operators. The second tier includes expert defined deductive rules and a flexible matching function. Experimental results provide evidence that two-tiered concept descriptions are not only simpler, but also have a better predictive power.

1. INTRODUCTION

Most methods of machine learning research assume that concepts are precise entities, representable by a single symbolic description. If an instance satisfies the given concept description, then it belongs to the concept, otherwise it does not. In contrast, most human concepts have a context-dependent meaning and lack precisely defined boundaries. The imprecision of the boundaries seems to have a logical rather than probabilistic character.

In order to handle such flexible concepts, machine learning systems need to employ richer concept representations than currently used. There have been numerous attempts to deal with representing of imprecise concepts, e.g. multiple-valued logic (e.g. Rine, 77), fuzzy logic (Zadeh, 74), and non-monotonic reasoning systems (e.g. Reiter, 87). Starting point of the research presented here is the idea of the *two-tiered concept representation* (Michalski et al. 86). In this representation the total meaning of a concept consists of two components, the *Base Concept Representation* (BCR) and the *Inferential Concept Interpretation* (ICI). The BCR defines the most typical properties of the concept. The ICI makes the boundaries of the concept flexible by describing allowed modifications of the concept's features in

---

This work was done while the first two authors were with the Artificial Intelligence Center, George Mason University.

different contexts.

Early ideas on learning two-tiered concept representations were presented in (Michalski 87) and closely related earlier work (Michalski et al. 86). An intriguing result of that research was that a concept description can be reduced to its most important components, without affecting its performance. This paper is an extension and continuation of these early ideas. Important novelties are the introduction of a general description quality measure, the use of both a rule base and a flexible matching function in the ICI, and the development of a heuristic search procedure for simplifying the concept descriptions. Simplifications are more selective. They also have a better heuristic justification in terms of their impact on the accuracy of the simplified descriptions, than the simple truncations described in (Michalski et al. 86). Empirical results discussed in sec. 5 show that simplification of concept descriptions by heuristic search, as proposed in this work, produces descriptions performing comparably well with the results of (Michalski et al. 86).

To demonstrate the methodology of learning two-tiered concept descriptions, we have implemented a system and then applied it to selected problems. Experimental evidence shows that by applying a two-tiered concept representation one can significantly decrease the complexity of a description, improve its comprehensibility and at the same time obtain better accuracy in recognizing new instances of the concept.

2. CONCEPT REPRESENTATION

The Base Concept Representation (BCR) is a disjunctive normal form which is called a cover; it defines the most typical properties of a concept in a simple and efficient form. A cover is a disjunction of complexes and a complex is a conjunction of selectors (Michalski 83). The Inferential Concept Interpretation includes a flexible matching function and a set of deductive rules.

The flexible matching function F is predefined and fixed during learning and testing. The value of F for an event e and a cover c is defined as the probabilistic sum of the values of F for its complexes, e.g. if c consists of a disjunction of two complexes *cpx1* and *cpx2*, then:

$$F(e, c) = F(e, cpx1) + F(e, cpx2) - F(e, cpx1) * F(e, cpx2)$$

F of event e and a complex *cpx* is defined as the average of the Fs for a conjunction of its constituent selectors, weighed by the proportion of positive examples covered by the complex:

$$F(e, cpx) = (\sum_i F(e, sel_i) / n) * \#cpxpos/(\#cpxpos + \#cpxneg)$$

where n is the number of the selectors in *cpx*, and *#cpxpos* (*#cpxneg*) is the number of positive (negative) examples covered by *cpx*, respectively.

F of an event e and a selector *sel* is defined by the degree of match between the event and the selector weighted by the coverage of positive and negative examples of the selector:

$$F(e, sel) = DegMatch(e, sel) * (1+(\#selpos/\#pos - \#selneg/\#neg))/2$$

where *#selpos* (*#selneg*) is the number of positive (negative) examples covered by the selector, respectively. *#pos* (*#neg*) is the number of positive (negative) examples, respectively. The degree of match depends on the type of the selector. It is either 1 (match) or 0 (no match) for nominal selectors. In

case of a linear selector, *DegMatch* inversely depends on the distance of the event from the selector, weighed by the cardinality of the range of the selector.

The ICI also includes a set of deductive rules, allowing the system to recognize transformed or special cases. In fact, the flexible matching is most useful to cover instances that are close to the typical case. For example, flexible matching could allow us to recognize a sequoia as a tree, although it does not match the typical size requirements, while deductive reasoning would be required to recognize a tree without leaves (in the winter time) or to include in the concept of tree some metaphorical meaning (e.g. a genealogical tree or a search tree). The deductive rules in the ICI are expressed as Horn clauses. Rules in the ICI may chain, but simpler deductions are preferred, in order to make the classification easy to understand for the users. Although the implementation supports recursion, non-recursive rules should be used when possible, and the number of rule activations should be limited.

An event can then be covered by a two-tiered description through the following three types of matching:

1. *Strict matching*: the event matches the BCR exactly
2. *Flexible matching*: the event matches the BCR through a flexible matching function
3. *Deductive matching*: the event matches the concept through deductive reasoning by using the ICI rules.

In sequel, we use the term *inferential matching* when an event can be matched either flexibly, or deductively.

3. QUALITY OF CONCEPT DESCRIPTIONS

Our objective is to obtain concept descriptions of good quality, so the notion of quality has to be introduced, and an operational definition usable in our system has to be given. In the presented method, the quality of a concept description is influenced by three basic characteristics: the accuracy, the comprehensibility, and the cost.

The accuracy represents the description's ability to produce correct classifications. A common way to prefer more accurate descriptions is to require that they be complete and consistent with respect to the training events (Mitchell 77, Michalski 80). Even if a description is incomplete and inconsistent, the number of positive and negative examples it covers provides important information for evaluating its quality.

Accuracy of a concept description depends on the typicality of the examples it covers. A good description should cover the typical examples explicitly, and the non-typical ones implicitly. Moreover, covering a typical *negative* example in the BCR is far worse for the accuracy of the description than covering a non-typical negative example. Generally, descriptions in which the typical events are covered by the BCR, and non-typical and exceptional ones are covered by the ICI, are preferred.

The comprehensibility of the acquired knowledge is related to subjective and domain dependent criteria. An important requirement of an AI system is that knowledge has to be explicit and easily understandable by human experts. This is important for improving or modifying the knowledge, and for

communicating with experts.

The cost captures the properties of a description related to its storage and use. Other things being equal, descriptions which are easier to store and easier to use for recognizing new examples are preferred. When considering the cost of a description, two characteristics are of primary importance. The first one is the cost of measuring the values of variables occurring in the description. In some application domains, e.g. in medicine, this may be a very important consideration. The second one is the computational cost of evaluating the description. Again, certain applications in real-time environment, e.g. speech or image recognition, may impose constraints on the evaluation time of a description.

These criteria need to be combined into a single evaluation procedure that can be used to compare different concept descriptions. A possible solution is to have an algebraic formula that, given the numeric evaluations of single criteria, produces a number that represents their combined value. Nevertheless, the meaning of this final number is hard to understand for a human expert. Secondly, this may force the system to evaluate all the criteria, even if it would be sufficient to compare two given descriptions on the basis of the most important one, if one is much better than the other. In order to overcome some of these problems, we use a lexicografic evaluation functional (LEF) (Michalski 83) that combines the above mentioned criteria. A precise quality measure based on these ideas is given in (Bergadano et al. 88) and has been implemented in the system described in this paper.

4. IMPROVING THE QUALITY OF A CONCEPT DESCRIPTION

Learning two-tiered concept descriptions is performed in two stages. In the first stage, a complete and consistent concept description is obtained from an inductive learning system. In our approach, we have relied on AQ15 (Michalski et al. 86), and INDUCE (Hoff et al. 82) to obtain such descriptions. The description generated in this stage, together with an empty ICI, forms the initial two-tiered description for the second stage. The second stage improves the initial description generated in the first stage. This process is guided by the description quality measure discussed in the previous section, and is implemented as a best first search, i.e., the descriptions of better quality are considered first. According to the nature of the quality measure, descriptions can be improved mainly by increasing their accuracy or by decreasing their complexity. For this reason the operators in the search simplify the BCR of a given description by dropping (truncating) some of its components or by modifying the argument of some predicate. This does not always result in a loss of accuracy, especially when measured on a testing set of new examples, since simpler features might be more stable and depend less on the set of training examples. The search process is described as follows:

Search space:	a tree structure, in which the nodes are two-tiered descriptions (BCR + ICI) of a given concept.
Operators:	selector truncation, complex truncation, referent modification.
Search strategy:	controlled by the quality measure.

The goal of this procedure is not necessarily to find an optimal solution, i.e. the description with the

highest quality, because this would require a combinatorial search. On the contrary, the system tries to improve the given concept description by expanding a limited number of nodes in the search tree and is guided by a heuristic measure. In the current implementation these heuristics are based on the coverage of the individual selectors and complexes.

The operators in the search correspond to generalizations or specializations of the BCR of a given description. In particular, selector truncation is a generalization operator, making the new BCR cover more positive and negative examples, while complex truncation is a specialization operator, making the set of examples covered by the modified the BCR smaller. Referent modification can be either a specialization or a generalization operator, depending on the type of modification that is being used and on the type of selector involved.

After a search operation is applied on the BCR of a description d, referred to as BCRd, a new BCR may be either more specialized or more general than BCRd. If the description is more specialized, some positive events previously covered by BCRd may not be covered any more. On the other hand, when a description obtained from BCRd is more general than BCRd, some new negative events, previously not covered by BCRd, may have been added. These events are treated as exceptions. Two types ICI rules, rules that cover a positive events otherwise left out of the BCR and rules that eliminate a negative event from the BCR, are used to handle these positive and negative examples. These rules are learned from the explanations for the exceptions. In the system, these explanations are provided by human experts.

5. EXPERIMENTS

We have run experiments with the system in three different domains: labor management contracts, congressional voting record, and breast cancer reoccurrence. For the labor management application, we have collected real data on 57 labor-management contracts negotiated in Canada in the last 18 months. Each contract is described by sixteen attributes, belonging to two main groups: issues related to salaries (e.g. pay increases in each year of contract, cost of living allowance, etc.), and issues related to fringe benefits (e.g. different kinds of pension contributions, holidays, vacation, dental insurance, etc.). We have run two experiments. In each experiment we were dealing with two descriptions: a description of a contract, and a description of a contract proposal deemed unacceptable by one of the parties involved in the negotiation. We have evaluated concept descriptions both on the training set (18 positive and 9 negative examples) and on a testing set (19 positive and 11 negative examples) of cases not previously seen by the system. The results of the two experiments are given in Tables 1 and 2.

Table 1 Labor Management Contracts (First Experiment): performance of the descriptions derived by AQ15 (11 complexes and 28 selectors).

		Correct	Incorrect	No_Match
Strict Match				
	Training Set	100%	0%	0%
	Testing Set	80%	17%	3%
Flexible Match				
	Training Set	100%	0%	0%
	Testing Set	80%	4-17%	3%

Table 2 Labor Management Contracts (Second Experiment): performance of the descriptions obtained using the search procedure. In the case of inferential matching, expert rules were used in combination with flexible matching.

		Correct	Incorrect	No_Match
Strict Match				
	Training Set	63%	0%	37%
	Testing Set	43%	3%	54%
Flexible Match				
	Training Set	85%	0%	15%
	Testing Set	83%	13%	4%
Inferential Match				
	Training Set	96%	0%	4%
	Testing Set	90%	10%	0%

In the first experiment, we have used the descriptions learned by AQ15 (Initial Description), given in Fig. 1, and in the second experiment we have used the description generated by the system, given in Fig. 2.

[duration ≠ 1] & [wage_incr_yr2 ≠ 3.0] & [holidays ≠ 10] v
[wage_incr_yr1 ≠ 2.0% v 2.5% v 2.8% v 3.0% v 4.0% v 4.5%] v
[wage_incr_yr1 ≠ 2.0% v 2.5% v 2.8% v 4.0%] & [wage_incr_yr2 ≠ 2.0% v 4.0%] v
[wage_incr_yr1 ≠ 2.0% v 2.5%v 3.0%v 4.0%v 4.5%] & [holidays ≠ 9] v
[wage_incr_yr1 ≠ 2.0%] & [vacation = above_average] ::> acceptable contract

[wage_incr_yr1=2.0% v2.5% v 4.0%]&[holidays=10] &[vacation=below_average v average]v
[wage_incr_yr1 = 2.0% v 2.5% v 3.0%v 4.0% v 4.5%] &[wage_incr_yr2 = 2.0% v 4.0%] &
[holidays = 10] & [vacation = below_average v average] v
[duration = 1] & [wage_incr_yr1 = 2.0% v 2.5% v 2.8%v 4.0%] & [holidays = 9] &
[vacation = below_average v average] v
[wage_incr_yr1=2.0%v2.5% v4.0%]&[wage_incr_yr2=3.0%]&
[vacation=below_average v average]v
[duration = 1] & [wage_incr_yr1=2.0%v2.5% v4.0%]&[vacation=below_average v average] v
[wage_incr_yr1 = 2.0%] & [wage_incr_yr2 = 3.0%] ::> unacceptable contract

---

Fig. 1. Labor Management Contracts: Descriptions Generated by AQ15

[wage_incr_yr2 > 3.0%] v
[wage_incr_yr1 > 4.5%] v
[holidays > 9] v
[vacation = above_average] ::> acceptable contract

[wage_incr_yr1 = 2.0% v 2.5% v 4.0%] & [holidays = 10] v
[wage_incr_yr2 = 2.0% v 4.0%] & [vacation = below_average v average] v
[holidays = 9] v
[duration = 1] & [wage_incr_yr1 = 2.0% v 2.5% v 4.0%] v
[wage_incr_yr2 = 3.0%] ::> unacceptable contract

Fig. 2. Labor Management Contracts: Improved Descriptions.

The improved descriptions are simpler than the ones generated by AQ15, and they represent the most important characteristics of the labor management contracts: a contract is acceptable when it offers a significant wage increase (the first two complexes in Fig. 2), or it offers many holiday days, or the

vacation is above average. The training events that were not correctly classified by the description, as it was modified step by step during the search, were analyzed by a domain expert, who provided deductive rules allowing the system to classify almost all the training events (one of them could not be explained by the expert). Following is one of those deductive rules obtained for the ICI of the contract concept.

[wage_incr_yr1 < 3.1%] & [wage_incr_yr2 < wage_incr_yr1] ::> unacceptable contract

The rule addresses the case of a contract with a low wage increase in year one of the contract, and an even lower increase in the second year. In those circumstances, the holiday and vacation offered do not matter: the contract is deemed unacceptable by the union.

The combination of the modified description and inferential matching (rules plus flexible matching) produces the best results (90% correct classifications). The power of the improved description is due to a combination of all three types of matching (strict, flexible and deductive), and all three contribute to the quality measure of a description as computed during the learning process. This represents a new feature of the system, since inferential matching is usually introduced only after the learning phase is completed (Bergadano, Giordana, Saitta 88; Michalski et al. 86).

The second application was concerned with the U.S. Congress voting record. We have relied on the same data set as used by (Lebowitz 87) in the experiments on conceptual clustering. The data represents the 1981 voting record for 100 selected representatives. The data set was split randomly into a training and testing set, with voting records of democrats entered as positive examples, and voting records of republicans entered as the negative ones. The goal was to obtain discriminating descriptions of democrat and republican congresspersons. Again, we ran two experiments as in the first application domain. The results are shown in tables 3 and 4.

Table 3. Congress Voting Record (First Experiment): performance of the descriptions derived by AQ15 (10 complexes and 32 selectors).

		Correct	Incorrect	No_Match
Strict Match				
	Training Set	100%	0%	0%
	Testing Set	86%	14%	0%
Flexible Match				
	Training Set	100%	0%	0%
	Testing Set	86%	14%	0%

Table 4. Congress Voting Record (Second Experiment): performance of the descriptions obtained with the search procedure.

		Correct	Incorrect	No_Match
Strict Match				
	Training Set	84%	0%	16%
	Testing Set	73%	4%	23%
Flexible Match				
	Training Set	100%	0%	0%
	Testing Set	92%	8%	0%
Inferential Match				
	Training Set	96%	4%	0%
	Testing Set	92%	8%	0%

The learning set contained 31 positive and 20 negative examples, and the test set contained 29 positive

and 20 negative examples. The obtained results are comparable with the other application: the description was simplified and the prediction power was increased. In fact, the AQ description contained 32 selectors compared to 21 in the improved description and the recognition rate moved from 86% to 92%. In the Congress application, though, the use of the deductive rules did not cause any performance increase. This can be explained by an already high percentage of correct classifications on the test set (92%). Nevertheless, the result must not be underestimated: although performance is the same, deductive matching is preferred over flexible matching. In fact, the examples that are correctly matched through some ICI rule, are actually explained by relevant domain knowledge, and not only on the basis of a knowledge-independent distance measure. This is important both to accuracy and comprehensibility of the modified description when evaluated with the deductive rules. The quality measure used in the system actually reflects this idea and scores higher when inferential matching is used.

Finally, we have also experimented with the same breast cancer reoccurrence data as (Michalski et al. 86). Our system had a distinct disadvantage on this data, since in the absence of typicality information all examples were assumed to be equally typical. Moreover, no deductive rules were available The following results were obtained on the testing set:

Table 5. Breast Cancer Reoccurrence (Michalski et al. 86).

	AQ15	best complex	two-tiered
# of complexes	22	2	12
# of selectors	102	7	52
performance	66%	68%	70.5%

All these experiments show how a two-tiered learning scheme allows a system to learn concepts from a small number of examples, and produces simpler but still accurate descriptions. The concept meaning is now divided into a cover, generated by the search, a flexible matching procedure, defined a-priori, and a set of deductive rules given by the expert on the basis of misclassified examples selected automatically during the search process.

6. CONCLUSION

In this paper, a system that is able to learn two-tiered concept descriptions has been described. Learning is viewed as a state space search guided by a measure of quality that is applied to concept descriptions. The operators in the search modify a given description by removing some of its components or by simplifying the referent of the selectors. The goal of the search process is to obtain simpler but still accurate descriptions. In this way, the comprehensibility and the predictive power of the acquired knowledge are improved. Some of the motivations behind the system come from previous work (Michalski et al, 1986), that produced some preliminary results in which the flexible matching function was applied during the recognition process. In the system presented here, a different flexible matching function is used. It is augmented by a set of deductive rules, describing symbolically possible

transformations of the concept description. Selector truncation and referent modification are applied automatically during a search process.

The research presented here is related to recent and important work in machine learning that investigates the effects of simplifying concept descriptions (e.g. Fisher, Schlimmer 88; Iba et al. 88). An advantage of the presented method is that it does not experience any major loss of coverage as a result of description modification. Other relevant work is concerned with the problem of pruning decision trees (Cestnik, Kononenko, Bratko 87). An important difference is lack of constraints on the part of the representation that is truncated when learning a two-tiered concept description. In post-pruning of decision trees, only paths ending in leaves may be truncated, which may improve the efficiency at the expense of the description quality.

The system was tested on three different domain applications. The experimental results that we obtained confirm the hypothesis that two-tiered descriptions can be more accurate and easier to understand. The ICI used in the experiments included a flexible matching function and a set of logical rules. The performance of the descriptions produced by the search process on the test set is influenced by the use of the inferential matching. This is due to the fact that ICI is used during learning, in order to choose and modify the best descriptions. This property represents an important difference between the presented system and previous approaches, that tend to apply flexible matching only after the learning process is completed.

Acknowledgements

This research was done in the Artificial Intelligence Center of George Mason University. The activities of the Center are supported in part by the Defence Advanced Research Projects Agency under grant No. N00014-85-K-0878, administered by the Office of Naval Research, and in part by the Office of Naval Research under grant No. N00014-88-K-0226. The first author was supported in part by the Italian Ministry of Education, and the second author was supported in part by the NSERC grant number A2480. The authors thank Zbig Koperczak for his help with acquiring the contract data and A. Giordana for useful criticism.

References

Bergadano, F., Matwin, S., Michalski, R. S., Zhang, J., "Measuring Quality of Concept Descriptions", Proc. Third European Working Sessions on Learning, pp. 1-14, Glasgow, 1988a.

Bergadano, F. , Giordana, A. , Saitta, L., "Automated Concept Acquisition in Noisy Environments", IEEE Transactions on PAMI, 10, 4, pp. 555-577, 1988.

Cestnik, B., Kononenko, I., Bratko, I., "ASSISTANT 86: A Knowledge-elicitation Tool for Sophisticated Users", Procs. of the 2nd European Workshop on Learning, pp. 31-45 (1987).

Fisher, D. H., Schlimmer, J. C., "Concept Simplification and Prediction Accuracy", Procs. of the Fifth Int'l. Conf. On Machine Learning, Ann Arbor, pp. 22-28 , 1988.

Hoff, B., Michalski, R. S., Stepp, R., "INDUCE-2 - A Program for Learning Structural Descriptions from Examples", TR 82-5, ISG, University of Illinois, 1982.

Iba, W., Wogulis, J., Langley, P., "Trading off Simplicity and Coverage in Incremental Concept Learning", Procs. of the Fifth Int'l. Conf. On Machine Learning, Ann Arbor, pp. 73-79 (1988).

Lebowitz, M., "Experiments with Incremental Concept Formation: UNIMEM", Machine Learning, vol. 2, No. 2, pp. 103-138, 1987.

Michalski, R. S., "Pattern Recognition as Rule-guided Inductive Inference", IEEE Transactions on PAMI, vol. 2, no. 2,3,4, pp. 349-361, 1980.

Michalski, R.S., "A Theory and Methodology of Inductive Learning", Chapter in the book "Machine Learning, an Artificial Intelligence Approach", Michalski, R. S., Carbonell, J. G., Mitchell, T. M. (Eds.), Tioga Pub. Co., Palo Alto, Ca, 1983.

Michalski, R. S., Mozetic, I., Hong, J.,Lavrac, N., "The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains", Proc. 5th AAAI, pp. 1041-1045, 1986.

Michalski, R. S., "Two-Tiered Concept Meaning, Inferential Matching and Conceptual Cohesiveness" , Chapter in the Book "Similarity and Analogy", Stella Vosniadou and A. Orton, (Eds), 1987.

Mitchell, T. M., "Version Spaces: An Approach to Concept Learning", Ph. D. dissertation, Stanford University, December 1977.

Reiter, R., "Non-monotonic Reasoning", in Annual Review of Computer Science, (1987).

Rine, D. C., ed., "Computer Science and Multiple-valued Logic. Theory and Applications", North Holland, 1977.

Zadeh, L. A., "Fuzzy Logic and its Applications to Approximate Reasoning", Information Processing, North Holland, pp. 591-594, 1974.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

FROM ETS TO AQUINAS: SIX YEARS OF KNOWLEDGE ACQUISITION TOOL DEVELOPMENT

John H. Boose, Jeffrey M. Bradshaw, Catherine M. Kitto, David B. Shema
Knowledge Systems Laboratory, Boeing Advanced Technology Center
Boeing Computer Services, P.O. Box 24346, Seattle, WA, 98124, USA

ABSTRACT

Two major forces brought about the development of the knowledge acquisition tools ETS (the Expertise Transfer System) and Aquinas: technology push and application pull. Ideas from many areas were gradually integrated to meet the growing demands of knowledge-based systems problems at The Boeing Company. This paper briefly traces the history of this development and the sources of many of our ideas, describes features and the reasons that they were added, and illustrates typical applications at each stage of evolution.

First a brief analysis of the forces that led to the development of our knowledge acquisition tools is presented. Then a history of tool growth is shown. Included are various features that were introduced and the types of help they provided. Typical applications at each stage of development are illustrated. Finally, we list the next hurdles to be overcome and show how we expect to overcome some of them in the next few years.

1. DYNAMIC TENSION: TECHNOLOGY PUSH AND APPLICATION PULL

At each phase in the growth of new tools, there should be a balance between technology push and application pull. Developers have ideas they feel might be useful for solving a future application problem - *technology push* - and applications present more immediate problems that need to be solved - *application pull*. Technology push tends to be farsighted and application pull tends to be shortsighted. When technology push is the only force, there is a danger that any tools developed will not be useful since they are not anchored in real problems, only a developer's vision of what the problems may be. When application pull is the only force, tools tend to be developed for special purposes and are hard to generalize to other problems. In such a demand-driven environment, revolutionary (or even evolutionary) breakthroughs usually do not occur. Ideally, both of these forces should act on a technology development project at the same time, leading to tools that are farsighted, general, and yet useful for a variety of problems.

Boeing's Advanced Technology Center provides a unique environment that fosters the interplay of technology push and application pull. The Center sponsored an Associates' Program that helped transfer artificial intelligence technology to the rest of Boeing. Associates spent one year at the Center, where they attended courses and developed prototypes to solve a specific problem in their home division. When the year's training was complete, Associates returned to their groups where they continued to develop and eventually field the applications. Over 95 Associates graduated from the program; the majority of them built knowledge-based systems. At the same time, the Center was working with universities to import advanced technology projects and was developing its own projects in the areas of vision, robotics, speech understanding, natural language, machine learning, and knowledge-based systems.

2. SEEDS: THE ARCHITECTURAL PROFILE SYSTEM

Our knowledge acquisition tools are rooted in Personal Construct Theory (Kelly, 1955) and related methods, especially techniques based on repertory grid elicitation and analysis. Boose first explored repertory grid techniques in 1976 while a student at the University of Maryland School of Architecture, working in the Computer Science Department's Artificial Intelligence group. He was trying to build a software system that would help people design their own homes. The first component, the Architectural Profile System, was supposed to find out how people felt about house environments in terms of textures, visual use of space, materials, utility, colors, overall design style (Cape Cod, Modern, etc.), privacy, use of glass, relationship to the outdoors, and so on. Once variables were identified, the system was to match the user's preferences to existing styles and canonical templates. Other components of the system would then help the user complete a detailed design.

Boose explored the psychometric literature and implemented a repertory grid elicitation program in Fortran on a Univac 1108, based on Kelly's description of the Role Repertory Grid Test. Kelly's original repertory grid was used in psychotherapy. Elements were people the client knew; triadic comparison of these people helped elicit important characteristics (constructs) that could show clients how they viewed themselves and others. In the Architectural Profile System, descriptions of environmental situations were used as elements (for example, "A place where you like to read," "A crowded place that you like," "A place where you feel lonely"). Triadic elicitation produced constructs about the built environment (for example, small space / large space, hard surfaces / soft surfaces, comfortable furniture / showy furniture). Kelly's original non parametric factor analysis was implemented to produce constellations of constructs; these constructs then fed a Lisp-based laddering process (Hinkle, 1965) in which questions about specialization and generalization were asked in an attempt to expand the constructs and link them into a network. Data were gathered from about 30 subjects who used the repertory grid elicitation program.

This was as far as the system progressed. The next steps would have been to gather data about different architectural styles and components based on the types of constructs that were elicited (some data were assembled from styles based on about 20 well known architects), build a starting set of templates indexed to these style data, link the laddered network to the index, and build a graphics interface to show floor plans, elevations, and descriptions retrieved while using the system.

3. KAQ: A FRONT END FOR TEIRESIAS IN EMYCIN

In the spring of 1983, Boose joined the Boeing Artificial Intelligence Center (part of the Advanced Technology Applications Division, later to become the Advanced Technology Center). The Associates' Program started that summer with about 12 students. Early problems faced by the Associates who tried to build expert systems provided application pull for the initial knowledge acquisition efforts at the Center.

The first knowledge-based system "shell" used was a version of EMYCIN with a graphic tracing interface that ran on the Xerox 1100 Lisp machine. This system, developed at Teknowledge, was called KS300. Part of TEIRESIAS (Davis and Lenat, 1982), an early knowledge transfer tool, was embedded in this shell.

TEIRESIAS in EMYCIN provided debugging help after an initial knowledge base had been entered. The system also provided an intelligent rule and attribute editor and an interviewing dialog to help the user construct an initial knowledge base. This combination provided an excellent rapid prototyping environment for structured selection problems that is still not equaled by most of today's knowledge-based system shells. Still, Associates using this environment faced problems that still challenge knowledge engineers:

1. How should the knowledge base be structured; in EMYCIN terms, how should the context tree be structured?

2. How should rules (or frames) be chained together to properly drive reasoning?

3. When uncertain knowledge is needed, how can certainty factors be controlled to improve knowledge base performance?

4. What are the variables (variously called attributes, traits, characteristics, parameters, concepts, and dimensions, or in EMYCIN, *PARMS*) that should be included in rules or frames? What are the potential values for these variables? Which ones are more important, and how does this affect the knowledge base structure and reasoning?

Since 1) EMYCIN helped build a simple starting context tree, 2) rule chaining in initial systems tends to be shallow, and 3) issues about uncertain knowledge tend to be uncovered later in the knowledge engineering process, identifying and using variables for rules were the first real problems an Associate faced.

Boose thought repertory grids might be effective in helping Associates identify variables for rules. If an initial conclusion or solution set could be provided, then triadic elicitation could be used to help an Associate's expert

generate an initial list of variables and values. Once these variables were identified, they could be used to help build rules in KS300. This use of repertory grids and, later, related methods, provided much of the technology push in 1983 and 1984.

An attempt was made to resurrect the Architectural Profile System on the Center's VAX but the Fortran had a different "flavor" and many of the old Univac library modules had no corresponding modules on the VAX. In July, 1983, Boose started building a triadic elicitation system in Interlisp on the Xerox 1100 Lisp machine. Because of the advanced programming environment on this machine, a basic grid system was written in about a month. This system was called KAQ (for Knowledge AcQuisition; usually pronounced QUACK, or DUCK as it was dubbed by a colleague). KAQ was a substantial improvement over the Architectural Profile System. It included pop-up menus and windows, graphics with selectable objects, trait value *ranges* (from 1 to 5) as opposed to binary ratings, and allowed the recording of trait weights. A Wood Advisor was developed to test the system as it was being built; it used about 15 wood types and ten attributes. One of the Associates, Art Nagai, was building a database management system advisor using KS300 and had gotten stuck. This was the first "real" problem KAQ attacked. The system seemed to perform reasonably well and offered Nagai new insights that he applied to his advisor. A series of demonstrations was given to colleagues, Associates, and frequent visitors who toured the Center.[1]

In September 1983, KAQ contained the following features (tools or features are listed with bullets; uses and advantages of the tool are shown in italics):

- o Solution listing
  Identify potential solutions to the problem
- o Triadic comparison of solutions
  Identify important discriminating traits (variables) for rules
- o Ratings and trait weightings (1-5)
  Get judgments; show operational view of traits based on concrete solution objects (elements)
  Elicit relative importance of traits
- o Graphic editors- labels, ratings, addition, deletion
  Easily change information associated with grid

At this point, an interesting collection of articles was found describing tools and methods for extending repertory grid analysis (Shaw, 1981). Boose began applying ideas from some of the tools, notably ENTAIL (Gaines and Shaw, 1981) and DYAD (Keen and Bell, 1981).

The article including ENTAIL sketched a method that used fuzzy logic for discovering *entailments* between values of the traits (we later came to call these entailments *implications*). These implications showed logical generalizations, using the grid data as a sample set. An algorithm was implemented based on ENTAIL that added graphic output to KAQ to show networks of implications. Experts using KAQ could agree or disagree with these generalizations. They were often surprised by implications that they subsequently realized were true. These could give users new insights about their problems by transforming the knowledge and changing the way it was viewed. This idea became an important aspect of future work as more analysis tools were added.

When experts disagreed with implications, methods were established for reviewing the ratings in the grid for errors, adding "exception" solutions to the grid to make the offending implications disappear, and using laddering to check the consistency of use of the traits across all the solutions (Boose, 1984, 1985, 1986a). These analyses were first carried out manually; later they were implemented in ETS. Implication generation and review were the first analysis tools for debugging the information in repertory grids. Implicit in the point of view that led to their development was that grid data would be used for problem-solving - a very different use than the one originally intended by Kelly.

Implications also proved useful for identifying knowledge at different levels of abstraction. They could, for example, point out "part of," "kind of," or subsumption relations. Implications could also show the relative independence of traits (the ENTAIL algorithm produced an entailment strength from 0.0 to 1.0). For some

[1]Frequent demonstrations were, and still are, a way of life at the Center. We estimate that we have given over 300 demonstrations of KAQ, ETS, NeoETS, and Aquinas in the past six years. These demonstrations and use of the tools by Associates and others provide valuable feedback for new applications, increased utility, system limitations to be overcome, and improvements to the user interface.

applications, it is important to make traits in rules as independent as possible; this technique provided a measure of trait independence.

Hinkle, one of Kelly's students, had used other interviewing methods to generate implications (Hinkle, 1965). He analyzed these implications for patterns that he termed parallel, reciprocal, orthogonal, and ambiguous. These patterns could point out inconsistency, ambiguity, and equivalence among constructs. This pattern analysis was implemented in KAQ (Boose, 1986a).

DYAD used an incremental interviewing method to develop a grid. This would be useful when an expert could not readily list a possible conclusion set for a problem, so it was also implemented in KAQ (Boose, 1986a). Another article mentioned the use of ratings N (for neither pole applies) and B (for both poles apply). Situations often arose in which both of these ratings would be useful, typically when an element was outside a construct's range of convenience, so this feature was also added to KAQ.

In summary, by early October 1983, the following features had been added to KAQ:

- o Implication analysis (fuzzy set generalization)
  - *Show trait value relationships at higher level of abstraction*
  - *Summarize grid information*
  - *Note "problems" and fix manually (bad ratings, exceptions, consistency)*
  - *Provide another view of data*
  - *Measure trait independence*
- o Implication Analysis and Examination
  - Agreement
    - *Is it obvious?*
    - *Is it new to the expert?*
  - Disagreement
    - *Are ratings correct?*
    - *Is there an exception?*
    - *Are traits being used consistently?*
- o Hinkle's implication analysis
  - *Point out relationships, especially problems (ambiguities and equivalences)*
- o DYAD dialog (incremental interviewing)
  - *Expand grid with important solutions and traits*
  - *Use if expert has no initial set of solutions*
- o N (neither pole applies) and B (both poles apply) ratings
  - *Note when a trait does not apply comfortably to a particular solution*

Did all of this help knowledge engineers? Art Nagai, for example, claimed that KAQ gave him significant help for advancing the Database Management System Advisor. Others who used KAQ said they received insights that helped them start prototyping or enhancing projects using KS300.

Other typical applications from this period include a jet engine diagnosis aid, a communication network bug analyzer, and a Seattle Japanese restaurant advisor. Most of the advantages and disadvantages of repertory grid methodology were uncovered in the next few months as KAQ was iteratively changed and improved. These are discussed below.

4. ETS: EMBEDDED REASONING AND INTEGRATED ANALYSIS

In October 1983 KAQ was renamed the Expertise Transfer System, ETS (paying homage to Davis, who coined the term "expertise transfer" in his thesis on TEIRESIAS).

Boose realized that KS300 rules could be generated automatically from implications. In an implication, a value or "pole" of a trait implies another value with a certain strength. By eliciting an additional piece of information from the expert (the "concept" or "scale" name for each trait), ETS could transform an implication directly into a KS300 rule:

Implication: MORE-EXPENSIVE implies FARTHER-AWAY (strength)

KS300 rule: If COST = MORE-EXPENSIVE
Then DISTANCE = FARTHER-AWAY (CF)

At first the fuzzy implication strength (from 0.0 to 1.0) was simply mapped directly onto KS300's certainty factor scale (from -1.0 to 1.0). Later, other methods (described below) were tried.

These rules pointed out relationships between traits, but another kind of rule, one that related values of traits to specific solutions, was needed. It turned out to be easy to generate these kinds of rules, too. For example, if the solution PARIS was rated 5 on the trait DISTANCE (CLOSER (1) / FARTHER-AWAY (5)), then the following rule could be generated:

If DISTANCE = FARTHER-AWAY
Then SOLUTION = PARIS (CF)

Another rule could also be generated:

If DISTANCE = CLOSER
Then SOLUTION = PARIS (-CF)

where the certainty factor (CF) is negative. Initially, both kinds of rules were generated internally in ETS in a simple frame format. Then several KS300 knowledge base files were examined to discover how rules, parameters, and control information were represented. In several days ETS was generating its first knowledge bases that could be loaded directly into KS300.

Many ad hoc methods for generating certainty factors were tried. The algorithm would be changed, ETS would generate a rule base for KS300, the rule base would be loaded, and test cases would be run in KS300 to see how well the results fit the expert's expectations. Art Nagai's patience as a database expert was invaluable. Eventually, an algorithm was used that employed the weight of the trait (assigned by the expert in ETS) and the relative strength of the rating (a rating of 5, for example, was stronger than a rating of 4). It could compute a maximum certainty factor based on the number of rules that were generated (Boose, 1986a). Although we did not have the resources to perform a formal study, KAQ knowledge bases loaded in KS300 produced reasonable results for several applications by several experts. The algorithm continued to receive minor changes until it was replaced in 1985 in NeoETS.

The use of rules generated from implications in KS300 produced interesting results. KS300 backchained on the rules to find the most general or controlling implications and asked about these first. Answering one question during a consultation could thus provide values for several traits, in some cases dramatically reducing the number of questions KS300 needed to ask. Sometimes this could cause problems, since the expert didn't always agree with all the implications ETS produced; the inclusion of such rules was made optional.

ETS helped people develop prototypes very quickly:

- o Experts found it easy to interact directly with the tool. In fact, almost all the experts who used KAQ and ETS wanted to come back and continue using it. The interface and underlying methodology were straightforward and transparent. Knowledge was entered in the expert's vocabulary. The analysis tools offered experts insights that they found interesting and surprising. The use of these tools at the beginning of a project seemed to short-circuit typical problems brought about by manual interviewing (expert insecurity, fear of exposing problem-solving information to others, and anxiety caused by not knowing what the process was all about). This general enthusiasm of experts for using KAQ and ETS was probably one of the most important factors in their success.

- o People didn't have to be experts to use ETS. People used ETS to help select workstations, classify personnel, buy cars, and improve their golf swings. ETS seemed to apply to any problem with a relatively small number of potential solutions. Even if the resulting consultation system wasn't perfect, ETS helped users derive insights from analysis tool use and helped them think about their problems in a structured manner.

- Knowledge engineers who watched experts use KAQ and ETS and examined system transcripts could quickly learn vocabulary, see potential solutions, learn the appropriate level of granularity for solving a problem, view the relative importance of initial traits, test a working prototype, and ask the expert further questions about the problem in a much more directed manner than was usually possible using regular manual interviewing methods. In fact, ETS was of great help to most Associates, since they generally had no background in formal interviewing techniques (as is still true for most knowledge engineers today).

- Rapid prototypes could be built in an hour or less. It usually took an expert a half hour or so to build an initial grid (typically from 6-by-6 to 10-by-10) and examine the implications. Then it took about ten minutes to generate rules, stop ETS, start KS300, and load the knowledge base. This minor delay was dealt with later by building a reasoning engine in ETS; see below. The expert could then build and save test cases to see how well the knowledge base performed. Problems were dealt with by reloading ETS, adding to or modifying information in the grid, and then retesting. This rapid prototyping capability led to the use of ETS for feasibility analysis of many different potential expert system projects. It was relatively easy to invite an expert over to the Center for a few hours and try out several project ideas.

- It was much easier to maintain information in a grid than in rules. Grids were a more compact form of representation and were easier to comprehend than a set of rules.

- Knowledge bases produced by ETS could be modified or combined with other knowledge base information in KS300; ETS could thus produce components of larger systems.

- ETS was a good training tool. It was easy to explain basic expert system concepts by example - ETS produced rules automatically and KS300 could run consultations. Managers and others involved in the potential project could see the working prototype and quickly get an idea about knowledge-based system technology and its problem-solving potential for a particular project.

We had also improved repertory grid methodology:

- In ETS, repertory grids were *operational*. Information contained in grids could be used by an inference engine to solve problems and help make decisions. The link to KS300 provided dynamic analysis, whereas previous methods provided only static analysis.

- In ETS grids had been applied practically to knowledge-based systems; grid techniques had been successfully applied to other areas such as training and market analysis.

- Having several kinds of analysis tools together in one system also offered significant advantages. Users could systematically transform grid information from one form to another to learn about its strengths and limits and measure different aspects of its problem-solving power.

There were, however, significant limitations to the methodology and its implementation in KAQ and ETS (Boose, 1984, 1985, 1986a). Some of these difficulties have been overcome or reduced in our subsequent work.

- The method seemed to be limited to *analysis* problems - problems for which the solutions could be enumerated comfortably. *Synthesis* problems, such as configuration, scheduling, and design for which solutions are typically built up from component parts, were difficult to address using grids. It also seemed difficult to use grid methods to elicit causal, procedural, or strategic knowledge. Often this kind of knowledge seemed necessary for controlling reasoning in complex problems.

- There was a limit to how much information could be comfortably represented in a single rating grid. One application included a 38-by-35 grid, but it was hard for the expert to use and manage, partly because some of the analysis tools took too long and partly because it was difficult to comprehend that much information at once. Some method was needed to decompose large grids into manageable, related subgrids.

- Representing solutions at different levels of abstraction led to trouble when using analysis tools and building rapid prototypes. This became clear as one of the first Associates tried to build a jet engine diagnostic aid. Parts and systems were listed as potential solutions (problem areas), and diagnostic symptoms were generated through triadic comparison. Unfortunately, solutions such as "spark plug" and "electrical system" both appeared in the grid. A spark plug was a component of the electrical system. What was really needed was a grid that represented high-level subsystems, individual subgrids that represented components of those subsystems, and so on. In this case, the "leaf" grids would contain field-replaceable components. If such a hierarchy of grids existed, traits on the top-level grid would be symptoms of problems in overall subsystems, symptoms one level down would relate to components of subsystems, and so on. This type of problem and the problem of large grids were the major motivations for building hierarchies in NeoETS, discussed below.

- Many types of problem-solving information did not fit comfortably into the types of traits used by ETS and other repertory grid tools (typically, ordinal traits with ranges from 1 to 5). In the Database Management System Advisor, for example, one important type of trait related database management system to hardware. Nagai needed several bipolar ordinal traits to represent this in ETS, such as "runs on VAX / doesn't run on VAX," "runs on IBM / doesn't run on IBM," "runs on CDC / doesn't run on CDC," and so on. It would have been much more convenient to combine all this information into one nominal trait, "computer type," whose values could be VAX, IBM, CDC, and so on. Then these actual values, or sets of these values, could appear in the grid, rather than numbers. This type of problem was a matter of convenience; it was possible but awkward to represent such information in ETS. Another problem with ordinal ETS rating scales was one of precision. In a composite parts advisor, an Associate wanted to relate critical temperatures in a manufacturing process to potential solutions. Two problems arose. There were more than five critical temperatures (where trait ranges were only from 1 to 5), so the Associate was forced to use multiple traits to represent these temperatures ("critical point A / not critical point A," "critical point B / not critical point B," and so on). More critically, the Associate needed to represent the exact temperature, not an ordinal representation of the temperature. These types of problems led to the introduction of nominal, interval, and ratio traits in NeoETS, as well as the ability to adjust the ranges of the traits.

- Consultations produced by ETS allowed users to enter only *soft* constraints. They could specify preferences about the solution set but could not indicate that a given trait should control a situation absolutely. For example, the user could specify that cost should be low on an ordinal scale (a preference of 1 on a LOW-COST (1) / HIGH-COST (5) trait) but high cost solutions might still be selected. The user might not be able to spend more than a certain amount of money, but this could not be specified. The expert needed a mechanism for adding these *hard* constraints; such a mechanism was added in Aquinas (described below). Interactions between traits couldn't be specified either, where knowing the value of one trait limited the values of another. For instance, knowing that cost should be low might imply other features of the solutions. This information couldn't be represented in an ETS grid, but, again, was later implemented in Aquinas.

- More tools were needed to help revise an initial rating grid to increase its problem-solving performance. More static analysis was needed to explore the grid's potential power, and a feedback link was needed from the inference engine so that dynamic analysis tools could directly examine performance results from real problems. Some of these problems, mentioned below, were addressed in Aquinas.

- Kelly had pointed out problems in the psychotherapeutic domain that related directly to the use of grids for problem-solving. In particular, it was hard to tell whether or not the elements and constructs sufficiently represented the necessary information. Issues of necessity and sufficiency continue to plague knowledge engineers as they seek to verify and validate knowledge-based systems, no matter what tools they use. We have begun to add testing and performance measurement features to Aquinas, but, clearly, this will remain a difficult problem.

Before the end of 1983, other improvements were made to ETS. ETS was trained to make knowledge bases for OPS5 running on our Center's VAX. This provided an inexpensive, easily accessible environment for running consultation systems produced by ETS. Further triadic elicitation and automated implication reviewing were also implemented. ETS then contained the following additional features:

- o Automatic rule generation
  - *Operationalize information in a repertory grid*
- o Rules produced in KS300 format
  - *Allow dynamic testing of grid knowledge - expert can agree or disagree with conclusions and improve knowledge*
  - *Significantly expand repertory grid methods*
  - *Implication rules in KS300 cut down number of questions asked*
  - *Potentially deliver knowledge bases produced by ETS on non-Lisp machine*
  - *Expand sphere of usefulness*
- o Manual test review
  - *Provide mechanical methods for improving behavior*
  - *Measure performance*
- o Rules produced in OPS5 format
  - *Provide VAX-based delivery of knowledge base produced by ETS*
  - *Provide delivery in multi-user time sharing environment*
- o Further triads
  - *Further improve knowledge base from "random" triads*
- o Automated implication review

In November 1983, Boose gave a talk to the local Puget Sound Association for Artificial Intelligence at the University of Washington. This was the first time ETS had been presented outside Boeing. He also started a paper that was presented next summer at the American Association of Artificial Intelligence conference in Austin (AAAI-84; Boose, 1984).

Early in 1984, an internal inference engine was implemented in ETS. This relieved the annoyance of stopping ETS and starting KS300 on the Lisp machine. It also allowed the addition of debugging aids such as consultation tracing and rank-order comparison. Experts would enter their expected results and ETS would compare the list with the actual results using a simple rank correlation measure. Thus, the performance of knowledge in the rating grid could be measured, and experts could see whether changes made to a grid resulted in overall performance improvement or degradation. New interesting triads could also be formed based on the worst misplaced solution in the set (Boose, 1986a).

We also implemented laddering and trained ETS to make knowledge bases for several other knowledge-based systems shells that were in use at the Center. This resulted in the following additional capabilities:

- o Embedded testing using internal inference engine
  - *Eliminate step of making KS300 knowledge base, switching partitions, and loading*
  - *Allow subsequent internal test reviewing*
- o Internal test review for rank order performance measurement and "worst misplaced solution"
  - *Allow problem-directed debugging*
- o Laddering
  - *Support ambiguity analysis in implication review*
  - *Further elicit more general or more specific traits*
- o Rules produced for LOOPS knowledge bases
- o Rules produced for S.1 and M.1 knowledge bases
- o Rules produced for TI Personal Consultant knowledge bases
  - *Support delivery in other hardware and software environments*

In the spring of 1984, Boose visited William Clancey at Stanford. They discussed ETS and Clancey's forthcoming paper on heuristic classification. Information from this discussion later formed the basis of the reasoning structure in Aquinas.

ETS was used in the first knowledge engineering course taught at the Center. A user's manual was produced for knowledge engineers and experts. ETS was installed on several other Xerox Lisp machines as Associates graduated and returned to their home groups. A non graphic version of ETS, running under ISI Interlisp, was also ported to the Center's VAX.

In September, 1984, Boose received a letter from Mildred Shaw inviting him to visit her and Brian Gaines at York University in Toronto and to present a paper at the Personal Construct Congress in Cambridge, England, in 1985. She had seen the ETS paper in the AAAI-84 proceedings and was interested in seeing ETS and talking with us. Boose and a colleague, Ray Allis, visited Toronto in October and gave several talks and demonstrations. Shaw and Gaines showed them PLANET, an Apple II-based repertory grid tool embodying most of the features they had written about. The discussions were very fruitful, and Boose, Gaines, and Shaw formed a relationship that led to, among other things, a series of knowledge acquisition workshops (discussed below).

PLANET had several useful analysis tools that Boose implemented in ETS. One of these was similarity analysis and review, where rows and columns in the grid were measured for similarity and difference. This was a powerful static analysis technique when applied to problem-solving information in ETS. For example, if two solutions were highly similar (the ratings in their columns matched significantly), then the inference engine would not be able to discriminate between them very well. The review mechanism allowed the expert to add traits in a directed manner to help solve these problems. This same process, when applied to rows, could help find redundant or highly dependent traits.

Another item contained in PLANET was a cluster analysis tool, FOCUS. A similar tool was implemented later in NeoETS to help experts decompose large grids into hierarchies.

It was common wisdom that only one expert should be used to build an initial expert system, but many Associates' problems demanded expertise from multiple sources. Sometimes knowledge about similar solutions was needed from more than one expert (multiple experts in the same domain); sometimes different areas of expertise from several experts needed to be linked together (multiple experts in different domains). One problem with combining knowledge from different experts was that information tended to be "averaged" across experts to reduce conflict, which results in the loss of special case-information. How could knowledge be represented and used from multiple sources?

In the fall of 1984, we combined expertise from multiple experts using ETS (Boose, 1986b). Expertise was first elicited individually from experts. Then ETS produced S.1 knowledge bases from the individual experts' grids, and the knowledge bases were manually pasted together and loaded in S.1. Each rule was produced with an additional screening clause that recorded which expert had contributed the knowledge. The resulting consultation system first asked the user for the set of experts to be considered; the user could weight experts by applying certainty factors to the response. Then S.1 backchained as usual to produce consensus consultation results. An S.1 mechanism was added to measure each expert's distance from the consensus, and the "most dissenting" expert's opinion was then displayed side by side with the consensus. This allowed the user to see a full range of opinion, not just the consensus average. Experts could also use their own sets of solutions and traits; S.1 would ask about each necessary trait during backchaining. To the extent that experts used similar vocabulary and problem-solving models, the user could more easily compare their individual results.

This method combined a delphi-like approach to gathering information independently with the rapid elicitation capabilities of repertory grids. Since each expert's knowledge was preserved, no special-case information was lost. Information could be gathered rapidly from a large group of experts. Later, we implemented this method directly in NeoETS (discussed below), along with analysis tools for comparing grids from different experts.This capability led to subsequent applications of NeoETS and Aquinas for group consensus and group decision making.

By the end of 1984, then, ETS also contained these features:

- o Similarity analysis and review (from PLANET)
  - *Measure static problem-solving potential of a grid*
  - *Provide specific help for increasing discriminating power of grid*
  - *Provide method for adding new solutions and traits in a directed manner*
  - *Measure trait subsumption and independence*
- o Multiple expertise knowledge bases built in S.1 and OPS5
  - 2200-rule AI tool advisor produced for OPS5/VMS
  - *Combine different sources on same or different topics*

In December 1984, Gaines and Shaw visited our Center and gave a colloquium on their views of knowledge engineering and personal construct methods. Plans were made for a knowledge engineering workshop at York University the following May.

By the end of the year, ETS had been used to produce hundreds of prototype knowledge-based systems and had begun to receive wide publicity within the AI community. Some of these applications included an AI library advisor, aircraft fault identifier, airplane design flutter analyzer, automated numerical control cutter consultant, flight controls diagnostic aid, ATLAS structural analysis advisor, bond durability consultant, business computing needs advisor, business graphics package consultant, composite materials advisors, documentation update consultant, energy control system model evaluator, failure modes and effects analyzer, finish advisor for materials, flight mode manager, materials technology advisor, mental health diagnostic aid, microcomputer needs analysis, molded rubber seal advisor, office automation system advisor, organization development intervention consultant, PAN AIR software aids, portfolio management aid, programming bug finder, programming language advisor, propulsion system advisor, rivet selector, robotic system application advisor, software management consultant, software release advisor, technical sales and services consultant, velocity analysis advisor, wine advisor, and word processing system consultant.

5. NEOETS: HIERARCHICAL REPERTORY GRIDS AND MULTIPLE DATA TYPES

In January 1985, Boose taught a knowledge engineering course that was attended by a University of Washington psychology graduate student, Jeff Bradshaw. Already familiar with repertory grids, Bradshaw became interested in ETS and started working at the Center in the spring. Bradshaw contributed key ideas to NeoETS in the areas of knowledge representation and reasoning.

By the fall of 1985, NeoETS had been born in response to many of the limitations of ETS, especially those related to large grids, problem decomposition, levels of abstraction, and multiple data types. In NeoETS, we attempted to augment repertory grids with new representations (hierarchies and multiple data types) and a fuller set of integrated analysis and reasoning tools. The most significant change was a de-emphasis of rule generation and an effort to deliver knowledge to end users within Aquinas.

The internal structure of ETS was removed and replaced several times to produce a system that could handle multiple hierarchical repertory grids, multiple data types, new analysis tools, and an internal reasoning engine for these new structures. We began referring to NeoETS as a *knowledge acquisition workbench.* The additional complexity created new challenges for effective elicitation, analysis, and testing of knowledge. Features of NeoETS are documented in (Bradshaw and Boose, 1986). They include:

- o New data structures
  - Four hierarchies - solutions, traits, cases, and experts
    - *Allow for complexity, multiple levels of abstraction*
    - *Handle multiple experts within one NeoETS knowledge structure*
    - *Partition knowledge around specific subproblems (cases)*
    - *Handle hierarchical relationships in any data type*
  - Four trait scale types - nominal, ordinal, interval, ratio
    - *Handle trade-offs between precision, convenience, conciseness, cost*
- o Reasoning engine based on the Analytic Hierarchy Process (Saaty, 1980)
  - *Serve as first attempt at reasoning with hierarchical structures*
- o Additional bridges produced for other shells (KEE; an internal C-based shell)

In May, a knowledge engineering seminar was held at York University, and later a new article about ETS provided more detail on the methods and their implementations. Work was also started on a book that was published early in 1986 (Boose, 1986a).

In the fall of 1985 we started a project with Seattle University's software engineering master's program that was to deliver a C-based version of ETS in the spring of 1986. This project was successfully completed, and we have continued with follow-on projects each year to enhance the utility of this tool. Currently, many of the useful tools in the Lisp-based version of Aquinas have migrated to the C-based version.

6. AQUINAS: A WORKBENCH FOR KNOWLEDGE BASE LIFE CYCLE SUPPORTAQUINAS DEVELOPMENT HISTORY

Further developments of our workbench were aimed at handling knowledge base life cycle problems, the user interface, delivery of knowledge bases (and of the workbench itself), and integration with other tools and systems.

1986

Early in 1986, David Shema and Cathy Kitto joined our project. Shema worked initially at improving the user interface and Kitto began work on a dialog manager to help the user handle the growing complexity of the workbench (Kitto and Boose, 1987). Shema eventually started work on validation and verification methods, implementing a case storage and batch-replay mechanism that could be used to measure knowledge base performance and even automatically improve information in rating grids (Shema and Boose, 1988).

The initial dialog manager was an expert system within Aquinas that used knowledge acquisition heuristics (about knowledge engineering, knowledge acquisition cycles in Aquinas, uses and limits of Aquinas tools, and idiosyncratic knowledge) and information about the current session and knowledge base complexity and completeness. Later, the dialog manager was expanded to help the user select an overall category for an application problem (Kitto and Boose, 1988). This would help the user decide 1) if Aquinas was appropriate for the problem and 2) what knowledge-based system tool might be useful. The dialog manager rank-ordered appropriate potential tools to use in Aquinas at each interaction cycle and suggested several to the expert. The dialog manager was difficult to maintain, as other team members constantly changed tools and the information that led to tool choices became outdated.

For reasons that should be apparent, we were unhappy with the name "NeoETS" and renamed the system Aquinas. From 1986 to the present, we added numerous additional tools and techniques (summarized in Boose, Shema, and Bradshaw, 1988); a few of the more interesting developments are mentioned here.

Bradshaw, dissatisfied with the use of the Analytic Hierarchy Process for reasoning (Saaty, 1980), implemented an extended version of Clancey's heuristic classification model (Clancey, 1986) that included a maximum-entropy-based method for propagating uncertainty (Boose and Bradshaw, 1987). An important aspect of this method included techniques for performing inductive inference in sparse hierarchies of grid information. Bradshaw also implemented such useful features as a cluster analysis based on FOCUS and the ability to represent discrete probability distributions in the cells of a rating grid.

Tools were developed, enhanced, or integrated to help experts structure information in hierarchies by noticing trait relationships and patterns in ratings (cluster analysis, laddering, trait value examination, Hinkle's pattern analysis, automatic or controlled inheritance, graphic cut and paste). Other tools were developed to analyze similarities and differences across groups of experts. Table completion and boundary analysis were developed to help check grids statically for consistency and completeness (Boose and Bradshaw, 1987).

In November 1986, the first AAAI-sponsored knowledge acquisition workshop was held in Banff, Canada, co-chaired by Boose and Gaines. This was the first in a series of annual workshops in Banff and Europe that provided a unique forum for leading knowledge acquisition theoreticians and practitioners (Boose and Gaines, 1989).

1987

In early 1987, Bradshaw implemented a form of ID3 (Quinlan, 1988) in the inference engine. It performed a cost-benefit analysis on repertory grids that allowed the inference engine to order consultation questions dynamically in a best-first manner.

Multiple expert capabilities mentioned above were added to the knowledge base structure and the inference engine. Users could select subsets of solutions and weight experts. Consultation results showed the consensus and the dissenting opinion. Tools for analyzing the differences and similarities between experts were added.

Boose documented the large number of applications and problems that had put KAQ, ETS, NeoETS, and Aquinas through their paces and reviewed uses of Aquinas to elicit procedural and strategic knowledge (Boose, 1988a). He implemented an on line help mechanism and documented the growing number of knowledge acquisition tools and techniques exposed at the knowledge acquisition workshops (Boose, 1988b).

Shema developed techniques for debugging knowledge bases and measuring knowledge base performance. Manual, directed, and automated methods were implemented in an effort to provide validation and verification tools for life-cycle support (Shema and Boose, 1988). Later trait sensitivity and solution stability analysis mechanisms were added.

In 1987, Bradshaw started a new project combining ideas from decision analysis and knowledge-based systems (Howard and Matheson, 1984; Holtzman, 1989). This led to interesting theoretical work on integrating repertory grids with knowledge structures used in decision analysis, such as influence diagrams, a kind of Bayesian network (Bradshaw and Boose, 1988; Bradshaw, Boose, Covington, and Russo, 1989a,b). Kitto also started her own project using KNACK, a knowledge acquisition tool originally developed at Carnegie Mellon (Kitto, 1988; Klinker et al., 1989).

1988

In 1988, Shema implemented a constraint mechanism that allowed experts and users to represent trait interactions and hard constraints (Boose, Shema, and Bradshaw, 1988). He added a consultation spreadsheet mechanism that allowed users to easily reason about hypothetical situations by temporarily modifying grid ratings and weights and immediately seeing graphic consultation results. Another feature added by Shema was a link to a graphic database so that images could be shown and used during consultations. He has been instrumental in improving the user interface with graphic display and input tools and screen organization devices.

More improvements were made to the internal reasoning engine. The algorithm was improved, and hooks were added so that experts and users could easily tailor certain steps in the inference process based on particular applications.

Initial work was performed on capturing explanations from experts during knowledge elicitation and on keeping a history of knowledge base changes. We believe that explanations will be useful as an aid for knowledge base maintenance and for allowing end users to review the expert's justifications.

Aquinas has long been used to "try out" different values of trait ratings, weights, and preferences to see how the recommendations are affected. Spreadsheet consultations were added in the Fall of 1988 which allow the experts or users to temporarily change values and observe how the consultation recommendations would vary.

Work has been done to integrate Aquinas into other environments and tools. In the summer of 1988, Envos, a Xerox spin-off company, announced that the entire Xerox-based programming environment was available on Sun workstations. Now, the full version of Aquinas is available to anyone in Boeing with a Sun workstation. The C-based version of Aquinas continues to be moved to different workstations. A recent version, MacQuinas, which runs on Macintosh computers, is being integrated with MacXotl (the Mac II version of Axotl, a knowledge-based decision analysis workbench implemented in SmallTalk-80) within an environment named "folie a deux" (Bradshaw, Boose, Covington, and Russo, 1989a,b). This will allow us to combine repertory grid and decision analysis techniques in one framework.

Many of the limitations of repertory grid methods in general and our implementations in particular have, at least in part, been overcome. However, several major problems are yet to be solved. These are discussed in the next section.

7. THE FUTURE: EMBEDDED KNOWLEDGE, SYNTHESIS PROBLEMS, AND LIFE CYCLE SUPPORT

Aquinas and Other Knowledge Acquisition Tools

Most knowledge acquisition tools derive their power from application domain knowledge. Figure 1 shows representative knowledge acquisition tools plotted on two axes: domain independence and problem type (discussed in detail in Boose, 1988b). Tools such as FIS and STUDENT are built for single-application systems. They are successful because their developers could embed very specific knowledge about the domain application, but they are difficult to generalize to new applications. Tools such as MOLE, MDIS, TKAW, and SALT derive power from knowing about patterns of knowledge types and problem-solving methods in certain classes of domains. Although they can be generalized within their target subdomains, developers find it difficult to generalize them to relatively unconnected domains.

On the other hand, tools like KSSO and Aquinas have no embedded domain knowledge. They are very general but lack domain power.

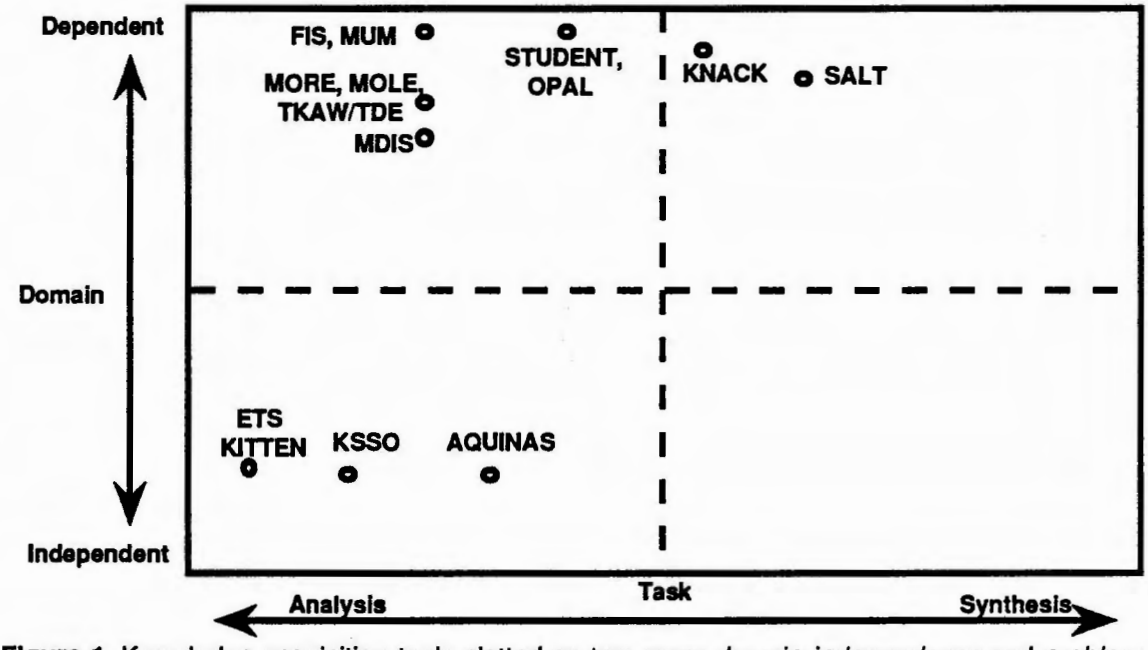


Figure 1. Knowledge acquisition tools plotted on two axes: *domain independence* and *problem type.*

One approach to improving Aquinas would be to develop domain knowledge templates that could help an expert start building a knowledge base (Bradshaw, Boose, Covington, and Russo, 1989b). One obvious area would be diagnostic problems where much is known about how to link symptoms, tests, costs of tests, and hypotheses. Loading such domain knowledge in Aquinas would significantly expand its problem-solving power.

Synthesis Problems

As can be seen in Figure 1, few knowledge acquisition tools have been successful at handling synthesis problems, and those that do are domain specific (Boose, 1988b).

We are trying to expand Aquinas to be able to handle simple synthesis problems such as configuration and process monitoring. For example, we are allowing users to put arbitrary computations in the cells of repertory grids in AQUINAS (spreadsheet-like computations, database access calls, functions that sample sensors) or to call specialized tools for alternatives generation and constraint satisfaction (Bradshaw, Boose, Covington, and Russo, 1989a; Shema, Covington, Boose, and Bradshaw, 1989).

Eventually, we hope to blend techniques from other tools (such as MDIS, Antonelli, 1983, and SALT, Marcus et al., 1985; Marcus, 1987) to help elicit knowledge for aspects of design problems such as:

- o Acquisition of causal models and design constraints.
- o Selecting alternative design concepts based on competing criteria such as reliability, maintainability, cost, and manufacturability.
- o Acquiring human expertise in validation and verification of designs.
- o Acquiring knowledge for design evaluation and change recommendation.
- o Acquiring knowledge for document generation and evaluation.
- o Capturing historical data in a "corporate memory" database to help solve future design problems.

We also intend to continue work on eliciting strategic and procedure knowledge to help control complex problems. While the dialog manager was successful in providing help to people learning Aquinas (Kitto and Boose, 1987), the rule-based format of its knowledge base was difficult to maintain. We are currently evaluating the use of repertory grids (Boose, 1988) and extended AND-OR graph representations (Bradshaw, Boose, Covington, and Russo, 1989a,b) to represent strategic and procedural knowledge.

Life Cycle Support

Figure 2 illustrates our intentions to increase Aquinas' knowledge acquisition power by embedding domain-specific knowledge and by augmenting Aquinas structures to handle synthesis problems.

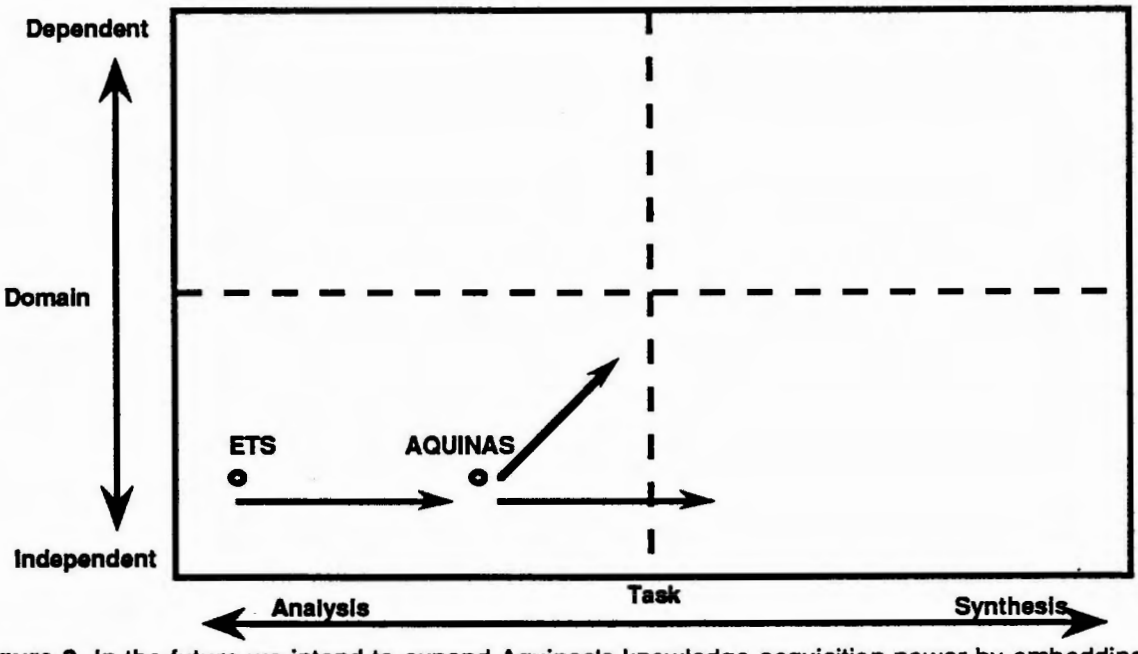


Figure 2. In the future we intend to expand Aquinas's knowledge acquisition power by embedding domain-specific knowledge and by augmenting Aquinas structures to handle synthesis problems.

In addition, we will attempt to support more of the knowledge-based system life cycle. We will concentrate on the areas of validation, verification, and maintenance of knowledge (Baum, Shema, and Boose, 1989).

In the near future, we will continue to build on Shema's work to measure and improve knowledge base performance, particularly in managing long-term maintenance of knowledge bases built by multiple experts. We will explore tools that tailor reports for maintainers after changes are made and perform sensitivity analyses to identify critical areas of knowledge. We may also implement TEIRESIAS-like features to model existing knowledge and identify inconsistencies when changes are made.

We will also explore tools for text analysis (Gaines, 1987) and improve our earlier work on explanation capture and playback mechanisms.

In the long term, we hope to provide full delivery facilities, including knowledge base partitioning and protection,and further facilities for eliciting and analyzing knowledge from multiple experts (elicit knowledge in parallel (on line) from several experts, analyze subsumption and overlap, support on line structured negotiation, perform intelligent knowledge base merging, discover the most important aspects of individual knowledge (special cases), and elicit and merge knowledge dynamically).

We are continuing our relationship with Gaines and Shaw, who are now at the Knowledge Science Institute at the University of Calgary, Alberta, Canada. Gaines and Boose have edited special knowledge acquisition issues of the *International Journal of Man-Machine Studies*, and are co-editing the Knowledge-Based System book series series for Academic Press, starting a new journal, *Knowledge Acquisition: An International Journal*, from Academic Press, and continuing to co-chair annual knowledge acquisition workshops in North America, Europe, and Japan.

8. CONCLUSION

Two major forces brought about the development of our knowledge acquisition tools : technology push and application pull. Ideas from many areas were gradually integrated to meet the growing demands of knowledge-based systems problems at The Boeing Company. This paper briefly traced the history of this development, and the sources of many of our ideas, described features and the reasons that they were added, and illustrated typical applications at each stage of evolution.

Building useful knowledge acquisition tools continues to provide exciting challenges to make knowledge-based systems and knowledge-based decision aids ever more practical. ETS and Aquinas have contributed significantly to the use of knowledge-based systems within The Boeing Company.

ACKNOWLEDGMENTS

Thanks to Miroslav Benda, Kathleen Bradshaw, Stan Covington, Brian Gaines, Sandy Marcus, Art Nagai, Peter Russo, Doug Schuler, Mildred Shaw, Suzzanne Shema, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. KAQ, ETS, NeoETS, and AQUINAS were developed at the Boeing Advanced Technology Center in Boeing Computer Services, Seattle, Washington.

REFERENCES

Antonelli, D. (1983). The Application of Artificial Intelligence to a Maintenance and Diagnostic Information System (MDIS), *Proceedings of the Joint Services Workshop on Artificial Intelligence in Maintenance*, Boulder, CO.

Baum, L. S., Shema, D. B., and Boose, J. H. (1989). Acquiring and Verifying Control Knowledge for a Blackboard System, *Proceedings of the 4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, in preparation.

Boose, J. H. (1984). Personal Construct Theory and the Transfer of Human Expertise, *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, Austin, Texas.

Boose, J. H. (1985). A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, *International Journal of Man-Machine Studies*, Vol. 23.

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

Boose, J. H. (1986b). Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain, *Future Computing Systems Journal*, Vol. 1, No. 2.

Boose, J. H. (1988a). Uses of Repertory Grid-Centred Knowledge Acquisition Tools for Knowledge-Based Systems, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 3, pp. 287-310.

Boose, J. H. (1988b). A Survey of Knowledge Acquisition Techniques and Tools, *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 1988,

University of Calgary SRDG Publications, Department of Computer Science; also in *Knowledge Acquisition: An International Journal*, in press.

Boose, J. H., and Bradshaw, J. M. (1987). Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1.

Boose, J. H., and Gaines, B. R. (1989). A Summary of the First and Second Knowledge Acquisition for Knowledge-Based Systems Workshops, *AI Magazine*, in press.

Boose, J. H. Shema, D. B., and Bradshaw, J. M. (1988). Recent Progress on Aquinas: A Knowledge Acquisition Workbench, *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November, 1988, University of Calgary SRDG Publications, Department of Computer Science.

Bradshaw, J. M., and Boose, J. H. (1986). NeoETS: Interactive Expertise Transfer for Knowledge-Based Systems, *Proceedings of the Symposium on Human-Computer Interaction and Cognitive Engineering* at the 1986 International Conference on Systems, Man, and Cybernetics, Atlanta, GA, October.

Bradshaw, J. M., and Boose, J. H. (1988). Decision Analytic Techniques for Knowledge Acquisition: Combining Information and Preference Models Using Aquinas, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.

Bradshaw, J. M., Boose, J. H., Covington, S. P. and Russo, P. J. (1989a). How To Do with Grids What People Say You Can't, *Proceedings of the 3rd Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, November 1988, University of Calgary SRDG Publications, Department of Computer Science.

Bradshaw, J. M., Boose, J. H., Covington, S. P. and Russo, P. J. (1989b). Folie a Deux: Integrating Aquinas, a Personal Construct-Based Knowledge Acquisition Workbench, with Axotl, a Knowledge-Based Decision Analysis System, submitted to the 3rd European Knowledge Acquisition Workshop, Paris, July, 1989.

Clancey, W. J. (1986). Heuristic Classification, in J. Kowalik (ed.), Knowledge-Based Problem Solving, New York: Prentice-Hall.

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

Gaines, B. R., and Shaw, M. L. G. (1981). New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems, in M. L. G. Shaw (ed.), Recent Advances in Personal Construct Technology, New York: Academic Press.

Gaines, B. R. (1987). Advanced Expert System Support Environments, in Boose, J. H. and Gaines, B. R. (eds.), *Proceedings of the Second Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October.

Hinkle, D. N. (1965). The Change of Personal Constructs from the Viewpoint of a Theory of Implications. PhD Dissertation, Ohio State University.

Holtzman, S. (1989). Intelligent Decision Systems, Reading, MA: Addison-Wesley.

Howard, R. A. and Matheson, J. E. (1984). Readings on the Principles and Applications of Decision Analysis, Menlo Park, CA: Strategic Decisions Group.

Keen, T. R., and Bell, R. C. (1981). One Thing Leads to Another: A New Approach to Elicitation in the Repertory Grid Technique, in M. L. G. Shaw (ed.), Recent Advances in Personal Construct Technology, New York: Academic Press.

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

Kitto, C., and Boose, J. H. (1987). Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 2, *International Journal of Man-Machine Studies*, Vol. 26, No. 2.

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.

Klinker, G., Genetet, S. and McDermott, J. (1989). Knowledge Acquisition for Evaluation Systems, *International Journal of Man-Machine Studies*, in press.

Marcus, S., McDermott, J., and Wang, T. (1985). Knowledge Acquisition for Constructive Systems, *Proceedings of the Ninth Joint Conference on Artificial Intelligence*, Los Angeles, CA, August, pp. 637-639.

Marcus, S. (1987). Taking Backtracking with a Grain of SALT, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 3, *International Journal of Man-Machine Studies*,

Vol. 26, No. 4, pp. 383-398; also in Boose, J. H., and Gaines, B. R. (eds.), Knowledge-Based Systems, Vol. 2: Knowledge Acquisition Tools for Expert Systems, New York: Academic Press, 1988, pp. 211-226.

Quinlan, J. R. (1988). Simplifying Decision Trees, in Gaines, B. R. and Boose, J. H. (eds.), Knowledge-Based Systems, Vol. 1: Knowledge Acquisition for Knowledge-Based Systems, London: Academic Press.

Saaty, T. L. (1980). The Analytic Hierarchy Process, New York: McGraw-Hill.

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, New York: Academic Press.

Shema, D. B., and Boose, J. H. (1988). 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.

Shema, D. B., Bradshaw, J. M., Covington, S. P., and Boose, J. H. (1989). Design Knowledge Capture and Alternative Generation Using Possibility Tables in CANARD, *Proceedings of the 4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, October, in preparation.

Stewart, V., and Stewart, A. (1981). Business Applications of Repertory Grids, London: McGraw-Hill.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

The Block Representation in Knowledge Acquisition for Computer Integrated Documentation

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

Paper for the 4th Knowledge Acquisition for Knowledge-Based Systems AAAI Workshop, Banff, Canada, October 1-6, 1989

Abstract

This paper describes an approach to the problem of Computer Integrated Documentation (CID) -- the problem of defining an adapted knowledge representation (KR) useful for capturing, storing, and using knowledge about an engineered system. The basic KR entity is a block of knowledge which includes five characteristics: a goal, actions, preconditions, abnormal conditions, and a context. Design knowledge capture is done using the block representation which leads to an analytical knowledge base usually available in the form of technical manuals. A block can be both the smallest entity of the knowledge system and the biggest. For instance, it can represent a page as well as a full manual. Furthermore, the block representation is very convenient for incrementally building operational knowledge. This later knowledge is called situational.

1. Introduction

1.1. The Documentation Problem as a Real World Example

Why Computer Integrated Documentation (CID) ? The complexity of modern engineered systems stresses the need for "good" technical and operational documentation systems. At present, technical and operational manuals are designed and developed to satisfy four goals: (1) storing systems architecture and internal functions, (2) storing performance and limitations (abnormal conditions) of these systems, (3) reuse of such information for operations, diagnosis, and maintenance, (4) reuse of such information as a basis for designing new systems.

Such documentation is very labor-intensive, requiring a great deal of expertise and development time. However, the main problem is that expert developers may no longer be available by the time a very large documentation project (such as the technical documentation of Aerospace systems) is nearing completion. Documentation is more useful during operations than before. Furthermore, most documentation systems (generally paper-based) are not entirely self-explanatory. Operators usually need help from documentation developers. At this point, developers may not be experts on the topic anymore. Thus, they

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

have to carry out a tremendous amount of problem solving activity to understand and retrieve the information needed.

With present documentation systems, it is difficult to integrate different sources of knowledge in a problem solving task. This integration can be done during the problem solving process (as is done now), i.e., human operators have to construct on-line dynamic connections between various available sources of knowledge. However, it can be facilitated if the knowledge is prepared in an appropriate form and if the connections between the various chunks[1] of knowledge are already programmed. As it is very difficult to design an appropriate connection network of knowledge, knowledge transformation will be necessary from a suboptimal knowledge base to a more operational one. To elicit such an operational knowledge, CID requires simulations or real-world experiments.

It is well known that technical paper documentation for an aircraft, for instance, weighs almost the same as the aircraft itself. Obviously, such documentation cannot be available on-board. Often, however, all this technical information is not necessary. In very specific cases which generally lead to incidents or accidents, "good" and complete technical information may be very useful for avoiding a catastrophe, e.g., the Chernobyl accident. Thus, computer documentation must be designed to be useful to the operator. At present, most operation manuals have been computerized and are available on-board on cathode raytube displays. They only include shallow operational knowledge sufficient for most operations. However, they do not provide assistance for lower levels of detail, which could be very useful in complex unexpected situations. In such situations, on future vehicles or complex systems, it will be necessary to get integrated answers inferred from lower levels of information. A shallow knowledge will not be sufficient.

A lot is known about how space systems work, but nothing is correctly formalized on how they will be used. A task may be very simple a priori, however as the environment is not predictable in advance, space systems cannot be fully automated. Thus, human operator(s) or decider(s) will take care of the overall management of the corresponding task using information which is more or less aggregated due to the fact that several sub-systems are more or less automated. Human intelligence and knowledge are used to compensate for the lack of a priori knowledge on the task. On the other hand, the probability of failure of a human operator is very high compared to equivalent procedural systems. For this reason, CID systems are defined as intelligent Operator Assistant (OA) systems which cooperate with operators to solve operational problems. They must be able to help astronauts in routine situations as well as in abnormal cases for following procedures, retrieving information, diagnosing faults, and in some cases planning activities. They are designed to work in a bounded safety domain and detect abnormal behaviors of the environment (i.e., for an OA system the environment includes the system to be controlled and the human operator). OA systems can also be used for implementing interactive incremental KA (see next section) techniques.

---

[1] The concept of chunk has been studied as a model of memory organization (Miller, 1956). A chunk is generally a symbol that designates a pattern of other symbols. Miller used this concept to prove that the span of the short-term memory is approximatively constant and independent of the complexity of items to be remembered. Chase and Simon (1973) also used the concept of chunk to explain why chess masters have an advantage over novices in reproducing chess position in memory. More recently Laird, Rosenbloom and Newell (1986) presented a general learning mechanism using the concept of chunking based on the SOAR architecture (Laird, Rosenbloom & Newell, 1985). The concept of chunk of knowledge, called "block" in this paper, is very similar to what has already been described by the above cited literature, I will give an alternative view to the chunk concept based on the concept of agent, i.e., an agent will be described as a network of blocks, each block behaving like a free-standing agent.

A CID system will include documents, e.g., reports, notes, memos, pictures, verbal explanations, and eventually video movies. A document is a particular block of knowledge which knows where to find help from other blocks if needed. Such information is accessible by messages. Furthermore, this representation must be appropriate to acquire substantive and strategic (Gruber, 1989) knowledge by using the CID itself.

1.2. SRAR Model

Generally, when the term "problem" is evoked in AI, the restriction to problem solving is understood. However, a problem is characterized by a problem statement and a problem solving process leading to a solution. It is common knowledge that a well stated problem is already half solved ! Moreover, when a beginner starts to learn a particular domain, he starts learning analytical knowledge which he will improve incrementally simply by augmentation of his knowledge base but also by transfer of various entities of his initial analytical sub-optimal knowledge base towards situational knowledge[1] better suited for routine use (more expert) (Boy, 1987). Results of this transfer lead incrementally to an operational knowledge.

The SRAR model (Situation Recognition and Analytical Reasoning) provides a formal framework for representing situational (problem statements or situational patterns) and analytical (problem solving resources) knowledge. It also provides a good framework for representing processes of learning by specialization/structuring: this model will be used for designing an incremental knowledge acquisition tool. At the beginning of the knowledge acquisition process, the system has a few small situational patterns (the system is inexperienced) and broad analytical knowledge which allows it to solve several problems in the given domain. As the knowledge acquisition process goes on, situational patterns (problem statements) become more complex and numerous, and at the same time the analytical knowledge becomes more structured.

When an OA system is implemented using the SRAR model, it can be used as a student, the operator being the teacher. It has a very analytical knowledge base at the beginning of its design and it will acquire more situational knowledge with practice. Analytical knowledge may come from first principles or from various episodes or scenarios. For instance, analytical knowledge is included in most manuals or books[2]. In the former case, the problem statement pattern matching will be done by instantiation and the resulting reasoning will be deductive. In the later case, it will be done by analogy. Both of these cases have to be taken into account for building an OA system.

---

[1] The quality of communication between two individuals relies on reciprocal understanding of each others internal model of thinking. For instance, a discussion between experts of the same domain is carried out in an operative language (Falzon, 1986) which is "very situational". In this case, experts have almost identical knowledge of the subject they are talking about, i.e., their internal models are nearly the same. Conversely, when a professor teaches, his internal model is very different from those of his students'. In particular, a professor has to "decompile" his own situational knowledge to make it understandable by novices. He will be said to be using an "analytical explanation" to make himself understood. This distinction between analytical and situational is not new. In his critique of Artificial Intelligence, Hubert Dreyfus (1979) claims that there is no expertise without situational knowledge. He claims also that it is very difficult to elicit and represent such knowledge in order to use it in computer programs. Following Hubert Dreyfus, existing expert systems can only represent "good" beginner knowledge, i.e., repeat the lesson of the professor or in the best cases, try to solve "very easy" academic exercises.

[2] It is generally very difficult to express situational knowledge using written support only. People mix text, drawings, and even gestures to express situational ideas. The main claim of this research is that, because they are good at simulation, computers can provide this kind of ability.

1.3. Knowledge Representation Issues

The SRAR model requires an appropriate knowledge representation (KR) which is tractable to both the computer and the user. Ochanine's operative image theory support the fact that users develop a restricted and distorted model of the object to process (Ochanine, 1981). This model is more suited to the task than a "true", "objective" representation (Falzon & Visser, 1989). However, a framework has to be built to acquire such users' operative models. This framework will be called knowledge representation following the AI terminology.

In this paper, it is claimed that the practice shows that situational (operational or operative) knowledge should be represented as an organization of independent knowledge entities which may be linked whenever it is needed. For this reason, the basic KR entity is a "*block*" of knowledge.

2. The Block Knowledge Representation

2.1. Definitions

Before starting to describe anything let us define the basic elements which are going to be useful. We will define intelligent behavior in the following manner: faced with a goal, a set of action principles is needed to pass from an initial situation to a final situation which satisfies that goal. In accordance with this model, one of the goals of AI is to build "intelligent" machines which interact with the environment. If these systems are so called "operator assistant systems", we also want to make sure that the end-user (operator) is effectively taken into account in the design process. Thus, an intelligent behavior should be one which integrates both human and machine intelligence to satisfy safety, workload, and performance criteria.

2.1.1. Agents

An agent[1] is a physical entity observable in the real world. Its potential internal mechanisms may be more or less complex. Their description may range from a simple list of properties to a very sophisticated model explaining its interaction with the outside world. Agents can be intelligent. In this case, they have goals and in each new situation they use procedures to satisfy those goals. Intelligent agents can be human or machine. The behavior of intelligent agents comes respectively from either natural intelligence or artificial intelligence.

2.1.2. Environment

In defining an agent, we are dividing the world into two parts: the agent and its environment, and we focus our attention on the interactions between the two. Thus, the environment of an agent includes all other agents (intelligent or not); notice that only some of them are relevant at any given time.

---

[1] The term agent has been chosen (instead of object) for all animate and inanimate entities because we will assume that each entity of the world is able to emit messages and receive messages from its environment, e.g., a table emits its color and is transformed by receiving new paint. In this case we will talk about a passive agent. It is more convenient to have the same name for further developments.

2.1.3. Messages

An intelligent agent is capable of anticipating the reactions of the environment as well as the consequences of its own actions. This anticipation will operate on the messages (Figure 1) coming from either the environment (external inputs or observations, e.g., m1 and m2) or the agent itself (internal inputs or inferred messages, e.g., m3 and m4).

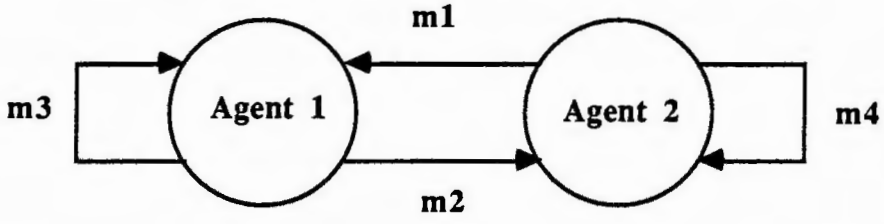


Figure 1. Messages and Agents

2.2. Description of the Block representation

Modeling an intelligent agent requires an appropriate knowledge representation (KR) which is tractable both for explaining human intelligence and implementing artificial intelligence. The KR currently in use was designed as a representation framework for operation manuals, procedures checklists, user guides (Boy, 1987) and other on-line tools useful for controlling complex dynamic systems (Boy & Mathé, 1989). Basically, this representation wants to handle operation procedures. We will represent the knowledge of an agent as a network of blocks of knowledge.

2.2.1. Block Basic Representation

The basic KR entity is a "block" which includes five characteristics: a goal, a procedure, preconditions, abnormal conditions and a context (Figure 2).

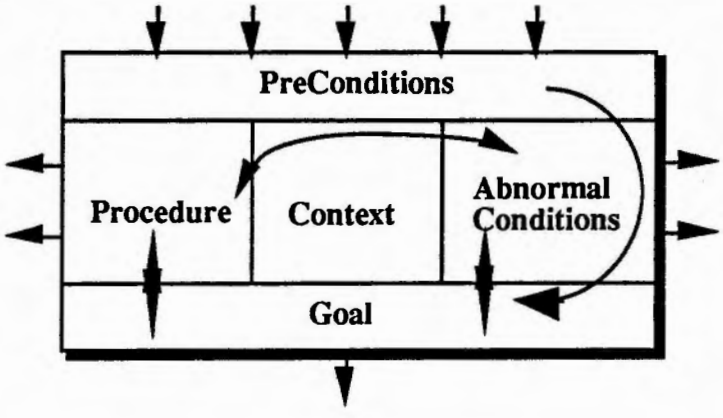


Figure 2 . Block basic representation.

A block cannot be activated if it is not in the right context or in an analogous context .

Preconditions (PCs) are conditions which lead to the execution of a procedure. A preconditions PC of a block of knowledge is represented as a conjunction of elementary preconditions PCi:

PC = ((PC1 & PC2 & ...), LogicalValue)

where:
(PC1 & PC2 & ...) represents a conjunction of elementary preconditions PCi,
LogicalValue = { TotallySatisfied, PartiallySatisfied, NotSatisfied }[1].

For each PCi, the representation template is:

PCi = (Predicate, LogicalValue)

where:
Predicate : e.g., On(A,Table);
LogicalValue= { Satisfied, NotSatisfied, Hypothesis }.

PC is TotallySatisfied if and only if:
for all PCi belonging to PC, such that PCi is Satisfied.

PC is not TotallySatisfied or PartiallySatisfied if and only if:
exist at least one PCi, such that PCi is NotSatisfied.

Goals are represented as follows:

G = (Name, LogicalValue)

where:
Name: is a string;
LogicalValue= { Suggested, NotSuggested, Reached, NotReached }[2].

Context are represented as follows[3]:

CX = (Name, LogicalValue).

where:
Name: is a string;
LogicalValue= { True, False }.

A *procedure* is represented as follows:

---

[1] { TotallySatisfied, PartiallySatisfied, NotSatisfied } means that the logical value of a precondition is either TotallySatisfied, PartiallySatisfied, or NotSatisfied. More generally, we will use the { . } notation to express mutually exclusive possible values.

[2] The initial value of a goal is generally NotSuggested. It can be suggested either from the outside of a block or when the preconditions are satisfied. When the procedure is completed the goal is Reached. If the procedure is interrupted by an abnormal condition then the goal is NotReached.

[3] The concept of context is loosely introduced here to define the concepts of strong and soft abnormal conditions, however it will not be develop extensively in this paper. It will be the topic of a next paper.

A = (A1 Δ A2 Δ ...)

where:

each action Ai is represented as follows:

Ai = { Name, Message, LogicalValue, ExpectedResults, AC(Ai) },
Name: is a string expressing the action meaning,
Message: is the message send by the agent executing the corresponding action;
LogicalValue = { Executed, NotExecuted },
ExpectedResults : preconditions from the environment as a result of the execution of the current action Ai,
AC(Ai) is a set of abnormal conditions applicable to Ai;

the operator Δ expresses a sequence and is defined as follows; if P is a procedure, i.e. a sequence of actions, and Ai an action, then P Δ Ai means that Ai will be executed after P with the following restriction: if expected results of Ai are already obtained from the application of P then Ai is no more necessary and we will have P Δ Ai = P.

Abnormal conditions are represented as follows:

AC = { AC1, AC2, ... }

where:

ACi = (Type, LogicalValue, Condition, Action)
Type = { WeakAbnormalCondition, StrongAbnormalCondition };
LogicalValue= { Violated, NotViolated }
Condition : expresses a condition on a message or a set of messages;
Action = (Send the-abnormal-condition-message to another-block).

Arrows represent messages. There are input arrows to the goal and to the preconditions of the block. This means that a block can be activated either by its goal being suggested or by receiving the preconditions. If the goal is not suggested and if the preconditions are all satisfied then the goal is suggested. If only some of the preconditions are satisfied and if the goal is suggested then the block may be "hypothetically" activated, i.e., all the unsatisfied preconditions are hypotheses.

There are output arrows from the procedure and abnormal conditions of the block. "Action messages" result when an action is executed. The corresponding agent effectively sends information to the environment. "Abnormal conditions messages" mean that when an abnormal condition occurs, the block effectively sends information to the agent (which the block belongs to): this information is typically "suggesting a new goal".

Internal messages are interpreted as follows. The arrow between preconditions and the goal means that preconditions (when all satisfied) can "internally" suggest the goal. The arrow between procedure and preconditions means that when an action is being executed a corresponding set of preconditions is loaded dynamically into the "short term memory" of the block (see section 2.3.2). Abnormal conditions could be general to the block or specific to a set of actions (meaning of the arrow between abnormal conditions and actions). Abnormal conditions are translated into preconditions in the short term memory (meaning of the arrow between abnormal conditions and preconditions). This means that preconditions can be static (the ones which have to be satisfied to start the procedure) or dynamic (the ones which have to be satisfied to execute the next action in the procedure).

2.2.2. Block Functional Representation

A block is composed of a virtual processor, a virtual long term memory and a virtual short term memory. The virtual long term memory includes all the elements of the five characteristics described in the basic representation. The virtual short term memory includes the current preconditions {PC1, PC2, ... }.The virtual processor uses the elements stored in the short term memory to deduce more information.

The execution of a block is started whenever its goal is suggested and/or its preconditions are totally or partially[1] satisfied. The execution of a block consists of performing its actions and controlling the satisfaction of the corresponding abnormal conditions. Abnormal conditions can be of two types: weak abnormal conditions which, if they occur, will cause an exit from the current block towards another possible block in the same context, and strong abnormal conditions which, if they occur, will cause an exit from the current block towards another possible context of blocks. A preference criterion mechanism is introduced to choose the best[2] abnormal condition handler if there are several abnormal conditions violated. This voting procedure defines the concept of strategy.

2.2.3. A block as a Problem

A block can be seen as a problem (Figure 3). Each problem has a name, at least as a label which can be handled to talk about it. In general, this name also expresses the goal of the problem. We will consider that the goal is actually the name of the problem. A problem is characterized by a statement and a solution[3]. The problem statement can be represented by a set of conditions to be satisfied for the problem to exist. In the agent representation, preconditions in a given context represent the problem statement. The problem solution may be explicit or implicit. If it is explicit, then it can be represented by a procedure of actions (instantiated plan of actions) applicable if attached abnormal conditions are satisfied. Obviously, if the problem solution is implicit, i.e., there is no instantiated plan of actions, then a set of actions and a set of constraints must be available to perform a plan. In this research effort, explicit problem solutions (i.e., procedures) are considered.

Problem:

Name	**(Goal)**
Statement	**(PreConditions, Context)**
Solution	**(Procedure, AbnormalConditions)**

Figure 3. Representation of a problem

2.2.4. Context of blocks

Description

Blocks are organized by context. In the figure 4, an example is given. Let {A, B, C, D, E, F, G} be a set of blocks having the same context CX. Messages between blocks are identified

---

[1] Criteria have to be introduced to define the term "partially satisfied".

[2] The definition of the "best" abnormal condition handler is application-dependent. The selection criterion may be defined with respect to the context, the task, real-time factors, constraints, safety, etc. Generally, people choose the most frequently used condition (Reason, 1986).

[3] The "solution" can be known or unknown. We will see the concept of solution as a dynamic process of refinement where a procedure is provided a priori but can be incrementally augmented and revised by encountering abnormal conditions in using it. Thus, we will define the concept "solution" as a (previously learned) procedure and a set of (incrementally acquired) abnormal conditions.

by numbers, from 1 to 13 in the given example. Block A is called the "root" of the corresponding context of blocks. By definition, when a violated abnormal condition leads to a message which directs towards a block which is in the same context, it will be called weak-abnormal condition, otherwise it will be a strong-abnormal condition. In general, messages can be:

- inputs into a context of blocks, e.g., message 1;
- outputs from a context of blocks, e.g., message 4;
- regular internal messages, e.g., messages 2, 3, 6, 9, 10;
- weak-abnormal condition messages, e.g., messages 5, 7, 11;
- strong-abnormal condition messages, e.g., messages 8, 13;
- loop messages, e.g., message 12.

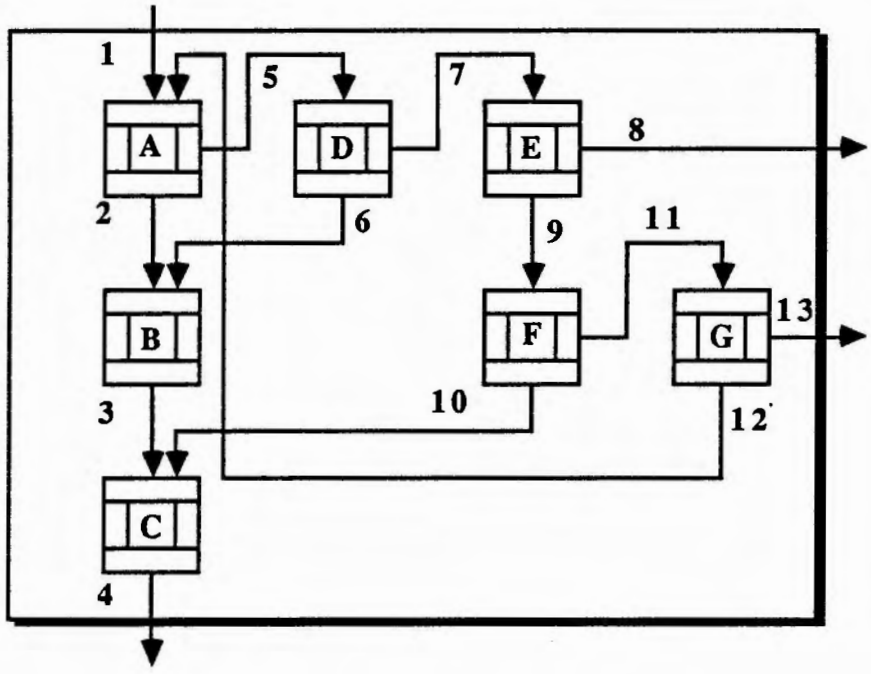


Figure 4. Context of blocks.

Zooming from a block to a Context of blocks

This representation is very convenient for representing tasks which have to be performed to accomplish given operations on an engineered system. Generally, this kind of work is done by operations engineers who are designing and developing operation manuals which have to be revised very often. This representation allows declarative programming by incremental decomposition. Knowledge design starts from very raw blocks described from a very high level (macroscopic) point of view. A block can be decomposed into a network of block which will inherit the context of the initial block (Figure 5). This will create layers of contexts of blocks which can be browsed easily. Furthermore, as a context of blocks is a block itself, the lowest grain of knowledge is being represented the same way as the highest. This property is very interesting from a KA point of view. For instance, in a document, a paragraph will have the same representation as a page, a chapter, a book, and a library.

Situational knowledge acquisition issue. An information gathering action is an active perception of a part of the environment. Such an action resulting from analytical reasoning can be "learned" and transferred into the situation pattern corresponding to the context of blocks where the initial block was included.

Procedure, Expected Results, and Abnormal conditions

In a given procedure, actions have expected results. Expected results can be either normal or abnormal. Possible abnormal results of the application of the given actions can be stored as abnormal conditions. Results of actions can either be checked during the execution of the actions or at the end of the execution of the corresponding block. In fact, satisfying a block goal expresses an execution in normal conditions, i.e., all the expected normal results happened and no abnormal results occurred. Conversely, violating block abnormal conditions expresses an execution in abnormal conditions (Note: Abnormal conditions include checking not only of abnormal action results, but also abnormal messages coming from the environment).

Each time an action is executed, the corresponding expected results are stored in the short term memory as preconditions.

2.2.6. An Appropriate KR for Knowledge Aggregation

Blocks can be very general, model-based, or very procedural. They constitute the analytical knowledge. Situation patterns are represented by a set of conditions to be satisfied. These conditions can be bound together by logical or mathematical operators. At the beginning of the knowledge acquisition process, situation patterns are very small, i.e., they include a few very simple conditions. As experimentation continues on the OA system, parts of some blocks can be transferred into situation patterns (Boy & Delail, 1988). More generally, parts of some blocks can be transferred into other blocks upstream in the hierarchy. For instance, if the procedure solving an abnormal condition is often the case, then it should be integrated (aggregated) into the procedure of the block(s) including this abnormal condition. A learning mechanism is provided in the next section of this paper to handle such a knowledge aggregation.

4. Exploration of an Academic Example

... using the Block Representation

A first mockup of the block representation is running on HyperCard. The "famous" Blocks World (which we will call the Cubes World to avoid any confusion with the Block representation) has been chosen as a very simple preliminary example to prove the presented concepts. The block represented on the card of figure 6 contains the four main components (Preconditions, Procedure, Goal and Abnormal conditions). In this example, the context will not be considered. The system and the user are considered as agents. What you see on the screen is the knowledge of the system, except for what you see in the center which is the external world fully independent of the knowledge of the system.

Figure 9 can be understood as follows. The system knows the goal to be reached, i.e., On(A,B) & On(B,C) & On(C,T)[1]. It expects everything from the external world (display in

[1] In this example, the predicate logic formalism will be used.

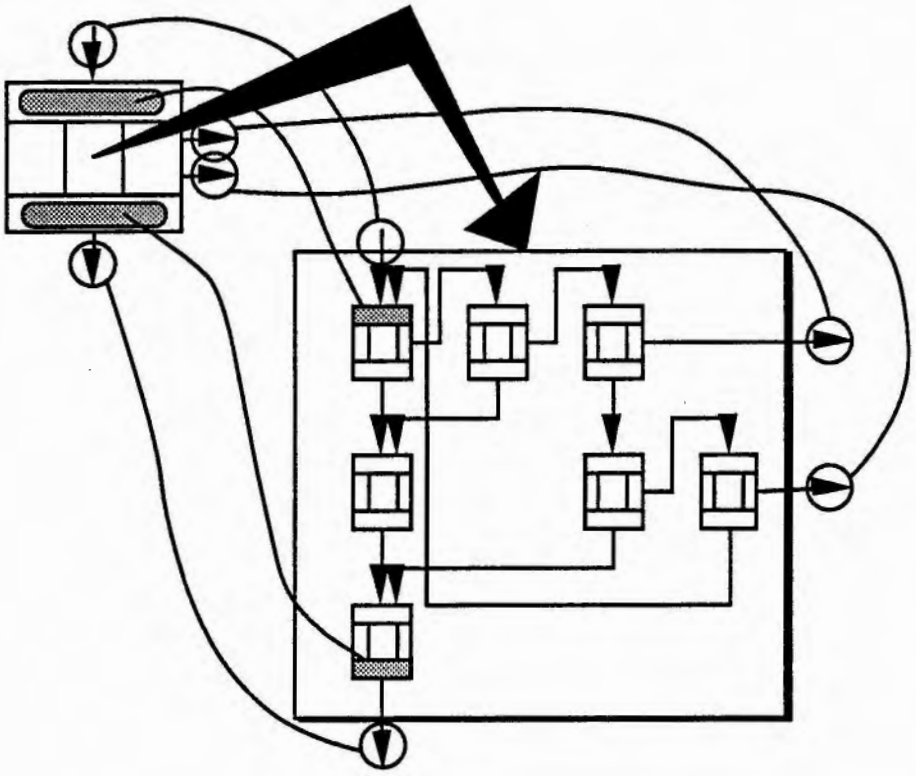


Figure 5. Decomposing a block.

Situation Patterns and Contexts of blocks

A situation pattern is a problem statement which leads to analytical reasoning on a context of blocks. As a context of blocks is also a block, the corresponding situation pattern is composed of the minimal[1] set of preconditions of the context of blocks. Blocks are then explored and executed to solve the problem stated by the situation pattern.

2.2.5. Properties of Actions

Action Classification

Actions can be classified as:

- information gathering actions (which are just providing information),
- actions that modify the external environment,
- cognitive actions (which modify the current internal state of the intelligent agent, these are actually problem solving actions).

Consistency checking issue. Depending on the grain level of the context representation, an action that modifies the external environment may cause a context change. Thus, this classification can be useful for consistency checking of the current knowledge base.

---

[1] "minimal" expresses the fact that, if a precondition is expressed in two different blocks of the same context, this precondition will be taken only once in the situation pattern.

the center of figure x). It does not have any action to execute, nor any abnormal condition. The user can tell the system what to do next. In other words, taking into account the external world, it is obvious that the only sensible action to be executed is MoveOn(A,B). The user selects this action and give it to the system agent (figures 10). Now, the system can solve the problem and reach the goal.

At this point, the problem is solved. But, let's take another initial condition such as the one displayed in the center of figure 11. The system automatically suggests to execute the action MoveOn(A,B). However, it is not possible to do that. The user informs the system that there is an abnormal condition which is, for instance, On(B,A), as show in figure 11.

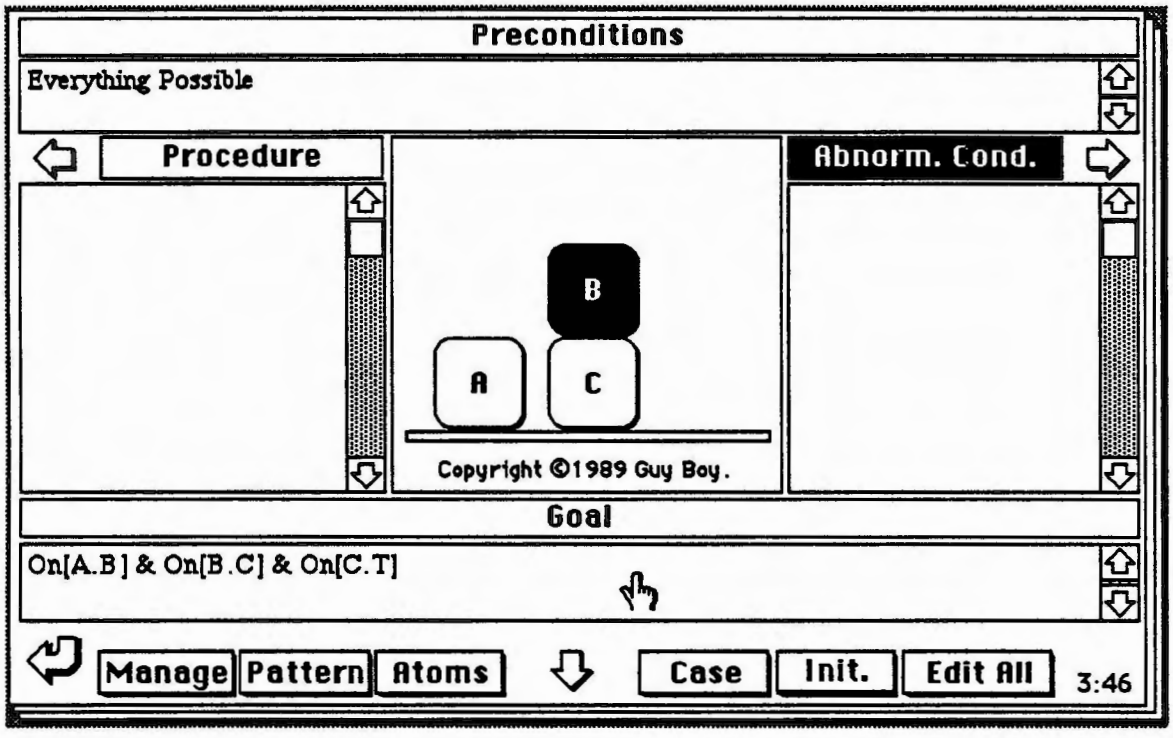


Figure 6. Block Representation and the Cubes World.

The user selects the left arrow to define a new block able to solve this abnormal condition. The user informs the system to move B on C in order to reach the goal On(B,C). But, when the system suggests to execute the action MoveOn(B,C), the user has to inform the system that there is another abnormal condition On(C,B) as show in figure 12.

Again, through the left arrow, the user creates a new block which will be able to solve this abnormal condition. The solution is displayed in figure 13. In this case, there is no more abnormal condition. The action is applicable to the real world. The result is displayed in figure 13.

The system then backward chain to the previous block which suggests to execute the action MoveOn(B,C). The result is displayed in figure 14.

Finally, the system proposes to execute the last action MoveOn(A,B), and the goal is reached.

Now, let assume that the abnormal condition On(B,C) is often the case. The user informs the system that this is a pattern (by selecting On(B,C) in the abnormal condition field and selecting the button "Pattern"). A new block will be created such as the one displayed in figure 15, and an abnormal condition has been automatically generated in the first block of the system (figure 16).

If the user informs the system that this new abnormal condition is also often the case, the previously build block becomes (Figure 17).

This new block includes the full sequence of action useful for reaching the goal from the precondition On(A,B) & On(B,C). This block does not include any constraint.

To formalize the pattern compilation mechanism, if there are two blocks such that:

block: B1

Preconditions:	PC1
Goal:	G1
Procedure:	P1 = (A_{11} Δ ... Δ A_{1n})
Abnormal conditions:	AC1

block: B2

Preconditions:	PC2 = AC1(A_{1i})
Goal:	G2
Procedure:	P2 = (A_{21} Δ ... Δ A_{2m})
Abnormal conditions:	AC2

and a particular abnormal condition AC1(i) of the list of abnormal conditions AC1 is often the case, then a new block B3 can be build from B1 and B2, such that:

block: B3

Preconditions:	PC3	= PC1 © PC2
Goal:	G3	= G1 © G2
Procedure:	P3	= (A_{11} Δ ... Δ $A_{1(i-1)}$ Δ A_{21} Δ ... Δ A_{2m} Δ A_{1i} Δ ... Δ A_{1n})
Abnormal conditions:	AC3	= ((AC1 - AC1(A_{1i})), AC2)

where:

Δ represents a sequence operator.

© represents the minimal conjunction which is defined as follows. The minimal conjunction is equivalent to the regular conjunction, except in the following generic case: if predicate P1 is more informative than predicate P2 then :

P1 © P2 = P1.

For example, On(A,T) is more informative than ¬ On(A,C), then:

On(A,T) © ¬ On(A,C) = On(A,T).

- represent the deletion operator. For example the result of AC1 - AC1(A_{1i}) is the set AC1 where AC1(A_{1i}) has been deleted.

Note that when a new block B3 is created from pattern compilation of the blocks B1 and B2 (as defined above), B1 is kept in the knowledge base and B2 is removed.

4. Application to CID

Our documentation system example is implemented using an hypertext system called HyperCard[1]. The hypertext technology allows non-linear browsing of the underlying documentation (Nelson, 1967; Englebart, 1963), e.g., cross-referencing is direct and easy. Embedded hypertext mechanisms improve search and exploration in a documentation. An hypertext system is defined as a set of nodes related between each other by links. Generally, nodes have an external representation on the screen, they can be stacks, backgrounds, cards, buttons or fields. Hypertext systems are very good tools for prototyping systems and developing user interfaces.

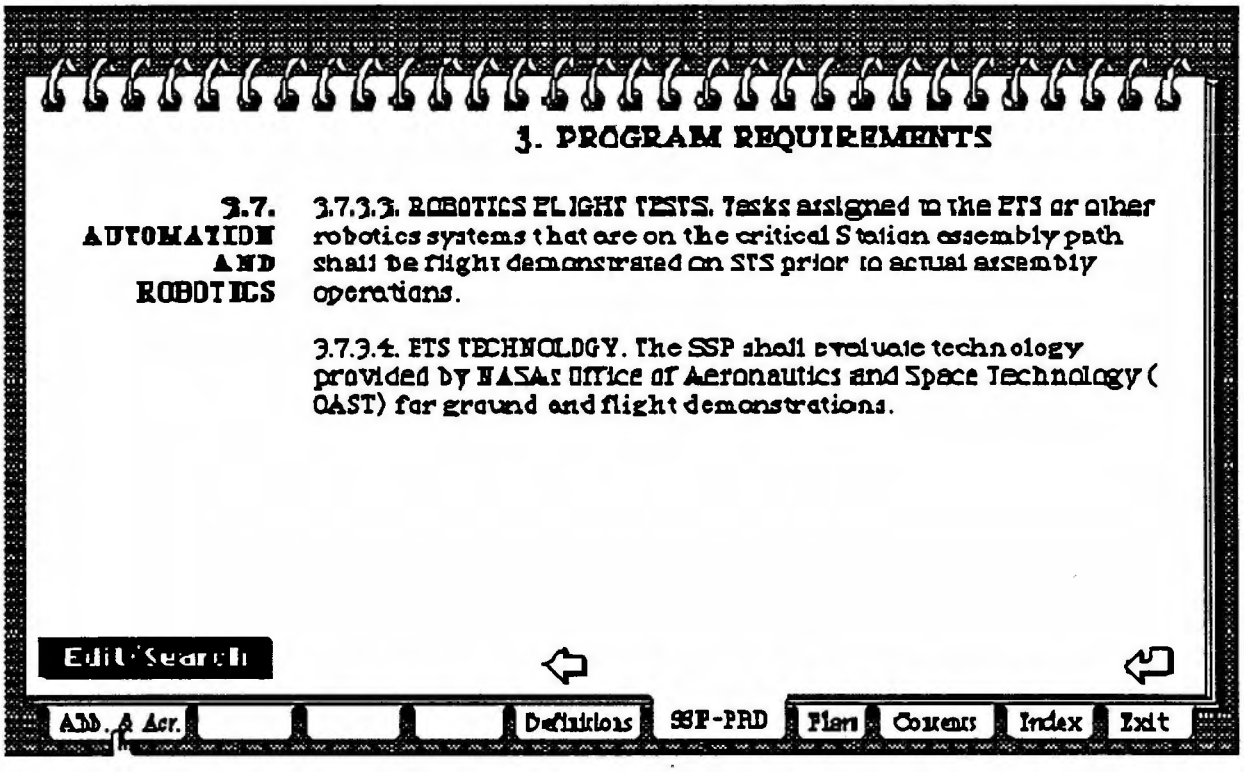


Figure 7. User selects index "Abb. & Acr." to find the abbreviation of "FTS"

At this stage of our CID project, we needed (1) to show how the end product would be, and (2) to face user with the system to acquire search strategies and develop appropriate indexes. Hypertexts have operational advantages such as: ease to trace references (this is particularly interesting for machine learning purposes), ease of creating new references, information structuring, global views, customizing documents, modularization of information, consistency of information, task stacking, collaboration between several authors (Conklin, 1987). We have chosen such a system for these reasons. However, hypertext systems have problems which are essentially (1) *disorientation*, i.e., the tendency to lose one's sense of location and direction in a nonlinear document; and (2) *cognitive overhead*, i.e., the

[1] HyperCard is a trademark of Apple Computer.

additional effort and concentration necessary to maintain several tasks or trails at one time (Conklin, 1987). For these reasons, the knowledge-based systems technology can be very helpful for taking, at least partially, this cognitive overhead to the user. That was another reason for developing the block representation on top of hypertext systems, i.e., nodes become blocks and links become messages.

Building new blocks from abnormal conditions discovery.

Let the user take a page of a manual[1] like the one displayed in figure 7. Let him/her search for the meaning of the three letters "FTS" included in the text.

By selecting the index "Abb. & Acr.", the user goes to the stack including abbreviations and acronyms. Then, the screen presented in figure 8 is displayed.

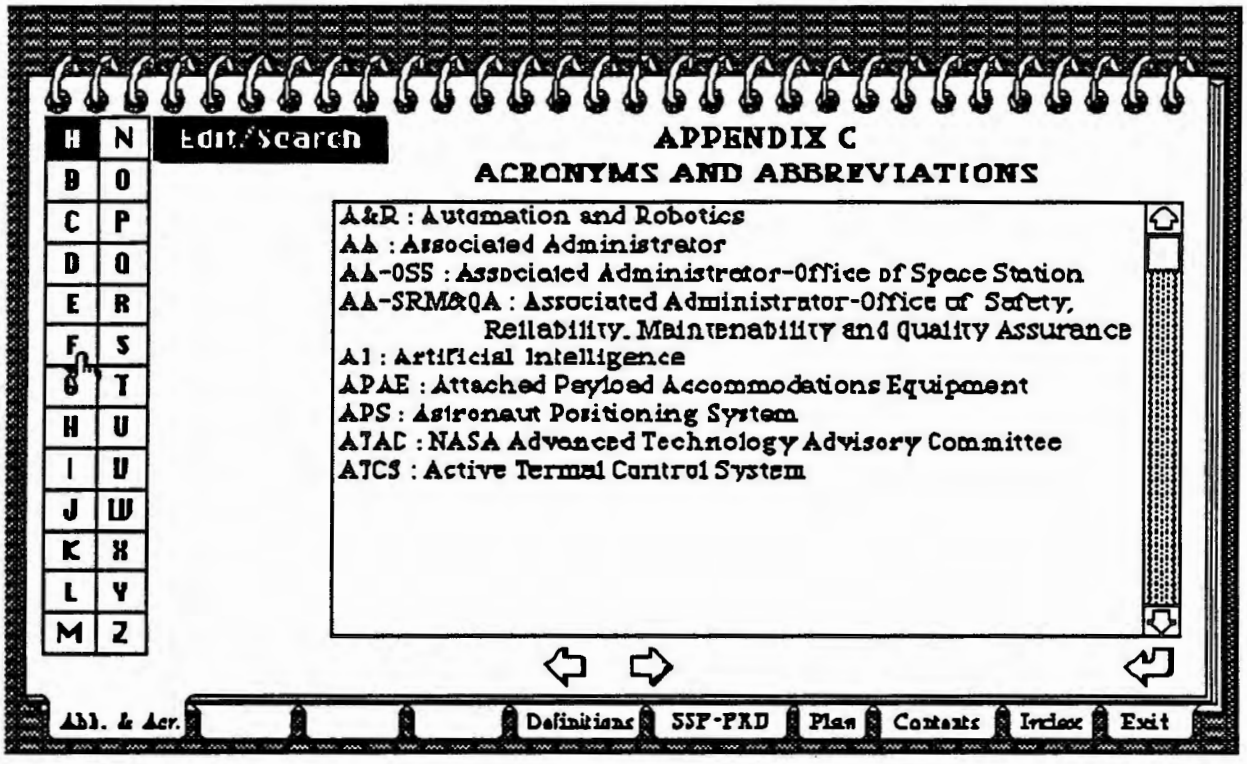


Figure 8. User selects index "F" to find the abbreviation of "FTS"

This display does not show any information about the abbreviation of "FTS". Then, the user selects the index "F". Therefore, the new display is presented in figure 9. As "FTS" is included in this display, the user's initial goal is reached.

[1] In this example, the document is provided in a stack called SSP-PRD, the abbreviations and acronyms in another stack called "Abbr. & Acr.", the definitions in another stack called "Definitions". Both in the abbreviations stack and in the definitions stack items are sorted by their first letter. A card has been designed for each first letter.

Let explain this sequence of operations. First, the system provided the user with a block called index "Abb. &Acr. " which allowed him/her to go to the first card of the Abbreviation stack. At this point, this block knew only these preconditions, goal and action, i.e.,

Block: User
Precondition: Right card of text
Goal: Get the abbreviation of the word "FTS"
Action: Select block "Abb. & Acr."

Block: "Abb. & Acr."
Precondition: user selects me
Goal: Get the first card of stack "Abb. and Acr."
Action: Go to the first card of stack "Abb. and Acr."

After that, the user selected index "F", which causes the display of the F-card of the corresponding stack. Finally, the user reached his/her personal goal. The interpretation in terms of block is:

Block: "F"
Precondition: user selects me
Goal: Get the card "F" of stack "Abb. and Acr."
Action: Go to the card "F" of stack "Abb. and Acr."

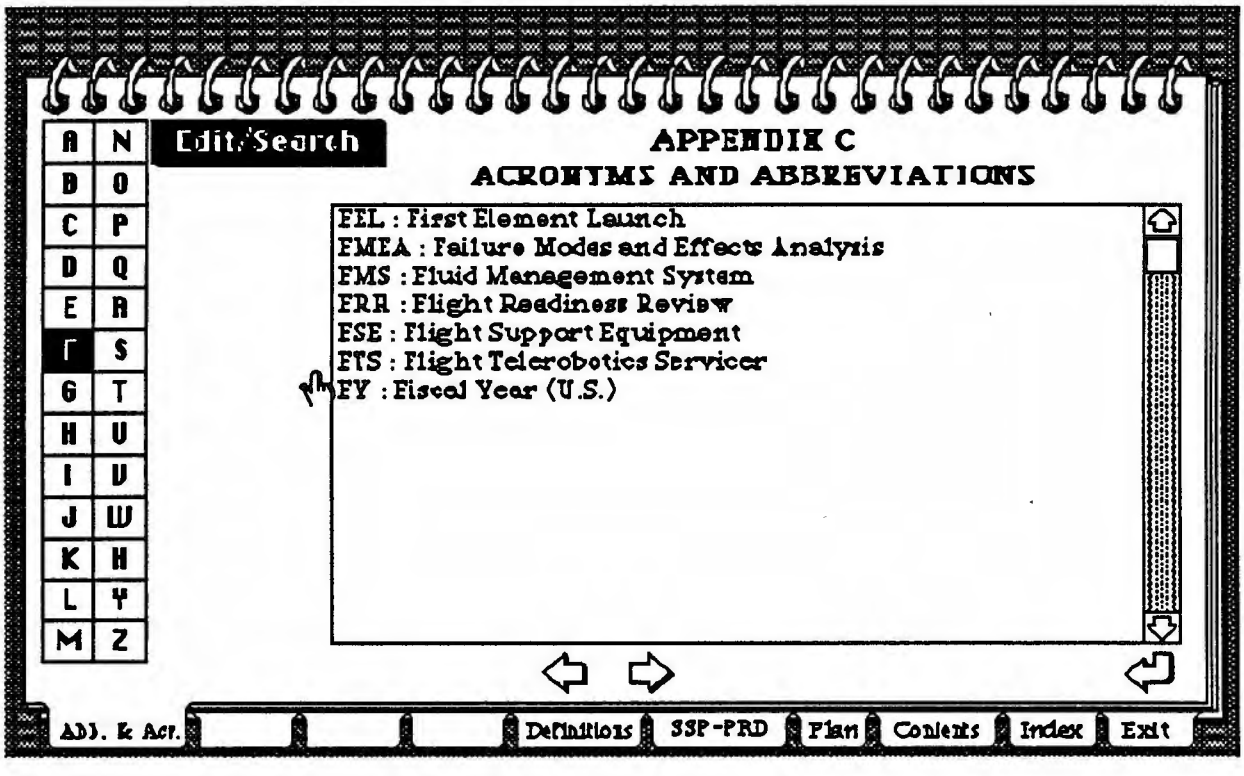


Figure 9. User found the abbreviation of "FTS"

Let build new blocks from this experiment. At this point, we know that the stack "Abb. & Acr." includes 26 cards corresponding to the letters of the alphabet. Each card contains a list of abbreviations and their meanings. We know also that the first letter of "FTS" is "F". Thus,

if we had selected "FTS" in the first place, we would have given the block "Abb. & Acr." with a complementary information which would be: (user selects the word "FTS"), the user would expect seeing the right card containing the abbreviation of "FTS". If we do not want to make severe modification in the pre-existing procedure used by block "Abb. & Acr.", this can be interpreted as an abnormal condition to the previous procedure. We then come up with the following block:

Block: User
Precondition: Right card of text
Goal: Get the abbreviation of the word "FTS"
Action: Select "FTS" in the text and select block "Abb. & Acr."

Block: "Abb. & Acr."
Precondition: user selects me
Goal: Get the first card of stack "Abb. and Acr."
Action: Go to the first card of stack "Abb. and Acr."
Abnormal condition: user selects also a word in the text

Block to be included in block "Abb. & Acr.": "Word selected"
Precondition: user selects a word in the text
Goal: Get the card (first letter of this word) of stack "Abb. and Acr."
Action: Go to the card (first letter of this word) of stack "Abb. and Acr.".

If we want the system to generate such block automatically, it has to know the explanation of the sorting mechanism in stack "Abb. & Acr.". The learning mechanism used above is generated by an abnormal condition introduced in block "Abb. & Acr." and a generalization on the (first) letter "F" of the (generic) word "FTS". In this paper, we will present a representation for generating abnormal conditions. The generalization mechanism will be presented in a following paper.

5. Relation to Other Work

Other work in this direction includes architectures which try to couple sensing to acting. In traditional AI, authors like Rosenschein (1986) and Kaelbling (1988) compile situation automata from specifications of what these automata are to know. Agre and Chapman (1987) in their implementation of a theory of activity advocate the importance of indexical over representational knowledge. Societies of block have also a similarity with the connectionist architectures (Rumelhart, 1986). Recently Nilsson (1988) described new techniques for organizing actions in dynamic environments in which the effect of actions and the state of the environment can not always be reliably predicted. His solution to the problem is based on hierarchical "action networks". In this representation, the entire network and each of its hierarchical components act like homeostatic servo systems attempting to maintain the environment in specific states. From this perspective, Nilsson's action networks and societies of blocks are almost identical in nature. I introduce here object and message-passing concepts to improve naturalness, implementation and connectivity of block and ("macro") agents themselves.

Other available knowledge representations are based of the definition of "blocks" of knowledge. Papert and Minsky proposed to substructure knowledge into "microworlds" (Minsky, 1975). Representing knowledge as a set of separate, simple fragments is the framework used by almost all AI researchers and cognitive scientists. Newell and Simon represents knowledge using "problem-spaces" (Newell & Simon, 1972). Considerable interest has been devoted to linguistic objects (Schank & Abelson, 1977).The representation

of "common sense" is a very difficult task because it confronts a type of knowledge which could be described as *compiled.* Common sense is a kind of envelope around a large number of facts, situations, and reasoning accumulated since the beginning of learning. The elicitation of this type of knowledge has always interested psychologists. In particular, Quillian (1968), Norman & Rumelhart (1975), Anderson (1976) were interested in psychological models of human memory. *Semantic networks* have progressively become established as functional representations in computer science (Woods, 1975). Notably, they have led to the notion of objects. Semantic networks were developed as descriptive models of the way in which the human brain makes *associations* between objects and concepts.

Designing a new knowledge representation (KR) seems perhaps pretentious in view of the considerable effort already invested in this field. To date, however, there is no system which successfully represents intelligence, integrating both perception and cognition. The neuro-network school says that intelligence is at the level of perception processes (Rumelhart et al., 1986). The cognitive school says that intelligence is at the level of reasoning (Laird, Newell, & Rosenbloom, 1984). From the point of view of learning, there are two directions: skill acquisition which is the process of compiling cognitive knowledge into associative knowledge which will be available at the perception level; and generalization which involves theory formation, the results of which will be used at a cognitive level. In both cases, induction could be a learning mechanism. Indeed, we can induce a skill from prior knowledge and experience (e.g., skill of riding a bicycle), or we can induce Newton's law from observing an apple falling from a tree. There is a need for a KR which is tractable and powerful enough for representing both skills and cognitive entities, in order to exchange information and knowledge between the two. The block representation is an attempt to integrate both forms of knowledge. Indeed, the abnormal-condition-based knowledge acquisition mechanism and the knowledge aggregation (pattern compilation) procedure provide an integrated framework towards this issue. This is an alternative way to integrating acquisition and performance systems (Rappaport & Gaines, 1988).

The SRAR model behind the present KR is quite different from "conventional" models of problem solving used in AI. Rasmussen (1986) defines human information processing as a process including a hierarchy of three levels: the *skill-based behavior* which is a set of well-integrated automatisms (kinds of stimulus-response processes), the *rule-based behavior* which corresponds to actual expert systems (IF-THEN rules) and the *knowledge-based behavior* which corresponds to real intelligence and education. When a skilled human operator performs routine tasks, he does not use the rule-based or knowledge-based level but the skill-based level. Human problem solving in routine tasks is guided by "patterns" learned from training and past experience. The operator's real expertise is not to reason analytically while he is under situational pressure. He is accommodating pre-learned procedures in real-time. In some rare cases, such local accommodations cannot work and lead to failure because the operator does not have a ready-to-use procedure. Even in this case, driven by a high workload, the operator will choose general heuristics which he accommodates as for specific situational patterns. This goal-oriented strategy, based on local accommodation of patterns is very efficient and has been observed in real world operations (flight test pilots, HORSES experiments, SAOTS experiments) (Boy et al., 1986, 1987, 1988).

6. Conclusion

The agent representation is designed as a representation framework for operation manuals, procedure checklists, user guides and other on line tools useful for controlling engineered systems. Basically, this KR belongs to (is an instance of) the object-oriented KR world. In this paper, I have described an agent representation suitable for capturing knowledge to be included in a CID. The agent representation allows the modeling of both design knowledge

and operational knowledge. The definition of a block satisfies the following five main requirements:

(1) it represents the main characteristics necessary for fast and efficient knowledge browsing and retrieval, i.e., this representation addresses the problem of how to increase the connectivity of a knowledge network in including abnormal conditions at the lowest level of grain;
(2) it is accessible from the top as well as from the bottom, i.e., sensitivity to goal and data is necessary in a reactive environment;
(3) it is incrementally modifiable, i.e., either it is easily expandable into several other blocks or able to exchange part of its knowledge with others blocks;
(4) a society of blocks[1], and at a macroscopic level a society of agents, has to be organized. The KR should allow several levels of organization, and for a given organization, re-organization should be easy. The block representation facilitates the KA process by the property of intrinsic recursiveness of blocks, i.e., representing the lowest level of grain in the same way as the higher levels;
(5) it includes the fact that a problem is composed of a problem statement and a problem solving process (which could be very easy, i.e., the solution is already attached as a procedure, or hard, i.e., the problem has to be solved using various techniques). The major assumption is that, at a very low level of grain, part of a problem can be represented by the simple paradigm (problem statement, problem solution) which is the macroscopic basis of a block. Problem solving is the trick which allows the problem solver to connect blocks in the most efficient way to find a solution. When the connectivity is expressed at a low level, inference is very fast, i.e., after executing a block, we know where to go next instead of searching for the next block to be executed. Improving system connectivity is a matter of both representation and learning.

Furthermore, interaction between a block and a human being should improve the common set of preconditions, i.e., the situation patterns in the SRAR model. In other words, a block can improve its expertise (in acquiring more sophisticated situation patterns) by interacting with a human user. Situation patterns (aggregated preconditions) define incrementally the human-computer interface.

Acknowledgements

Thanks to Philippa Gander for many useful comments on this paper at extremely short notice. Nathalie Mathé and I developed premises of these ideas together when I worked in the AI Research Group of ONERA/CERT. Thanks Nathalie for the careful review of this paper. Thierry Caminel deserves special acknowledgements for an early implementation of these concepts at Dialogics in Toulouse. I am very grateful to David Thompson, Catherine Baudin, Tom Gruber and Alain Rappaport who gave me some very purposeful comments on this knowledge representation.

References

Agre, P., & Chapman, D., (1987), "Pengi: An Implementation of a Theory of Activity", *Proceedings of AAAI-87*, Morgan Kaufmann Publishers, San Mateo, CA.

Anderson, J.R., (1976), *"Language, Memory and Thought"*, Hillsdale, Erlbaum, NJ.

[1] This idea of "society of blocks" is inspired by Marvin Minsky's "society of minds" (Minsky, 1985).

Boy, G.A. (1986), "An Expert System for Fault Diagnosis in Orbital Refueling Operations", *AIAA 24th Aerospace Sciences Meeting*, Jan., Reno, Nevada.

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

Boy, G.A., & Rappaport, A. (1987), "Operator Assistant System in Space Telemanipulation: Knowledge Acquisition by Experimentation", *ROBEX*, Pittsburgh, USA, Ju. 4-5.

Boy, G.A., & Delail, M., (1988), "Knowledge Acquisition by Specialization-Structuring: A Space Telemanipulation Application", *AAAI-88, Workshop on Integration of Knowledge Acquisition and Performance Systems*, St Paul, Minnesota, USA.

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

Chase, W.G., & Simon, H.A., (1973), "Perception in Chess", *Cognitive Psychology*, 4, pp. 55-81.

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

Dreyfus, H.L., (1979), *"What Computer Can't Do, The Limits of Artificial Intelligence"*, Harper and Row, Pub., Inc., New York.

Engelbart, D.C., (1963), "A Conceptual Framework for the Augmentation of Man's Intellect", in *Vistas in Information Handling*, Vol. 1, Spartan Books, London.

Falzon, P., (1986), "La communication Homme-Machine: Les Strategies de Dialogue et le Langage de l'Interaction", *Proc. Workshop on Human Machine Interaction and Artificial Intelligence in Aeronautics and Space*, G. Boy Ed., ENSAE/CERT, Toulouse, Oct. 13-14.

Falzon, P., & Visser, W., (1989), "Variation in Expertise. Implications for the Design of Assistance Systems", *Third International Conference on Human-Computer Interaction*, Boston, Elsevier Sc. Publishers.

Gruber, T., (1989), Automated Knowledge Acquisition for Strategic Knowledge", Report no. KSL 89-12, Knowledge System Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, to appear in a special issue of *Machine Learning* on automated knowledge acquisition.

Kaelbling, L., (1988), "Goals as Parallel Program Specifications", *Proceedings of AAAI-88*.

Keong, G., & Woo, C.C., (1988), "An Implementation of an Intelligent Communication Tool for Supporting Organizational Work", *Report* of the Management Information System Division, Faculty of Commerce and Business Administration, The University of British Columbia, Vancouver, BC, Canada.

Laird, J.E., Newell, A., & Rosenbloom, P.S., (1984), "Towards chunking as a general mechanism", *Proceedings AAAI-84*, Austin, Texas.

Laird, J.E., Newell, A., & Rosenbloom, P.S., (1985), "Soar: An Architecture for General Intelligence", *Artificial Intelligence*, .

Laird, J.E., Rosenbloom, P.S., & Newell, A., (1986), "Chunking in SOAR: The Anatomy of a General Mechanism", *Machine Learning*, Kluwer Academic Publisher, The Netherlands.

Miller, G.A., (1956), "The magic number seven, plus or minus two: Some limits on our capacity for processing information", *Psychological Review*, 63, pp. 81-97.

Minsky, M., (1985), *"Society of Minds"*, Touchstone Books, Published by Simon & Schuster Inc., New York.

Minsky, M., (1975), "A Framework for Representing Knowledge", in P. Winston Ed., *The Psychology of Computer Vision*, 211-277, New York: McGraw Hill.

Nelson, T.H., (1967), "Getting It Out of Our System", *Information Retrieval: A Critical Review*, G. Schechter, Ed., Thompson Books, Washington D.C.

Newell, A., & Simon, H., (1972), *"Human Problem Solving"*, Prentice-Hall, Englewood Cliffs, N.J.

Nilsson, N.J., (1988), "Action Networks", Computer Science Department, Stanford University, Stanford, CA 94305.

Norman, D.A., & Rumelhart, D.E., (1975), *"Explorations in Cognition"*, Freeman, San Francisco.

Ochanine, D.A., (1981), "L'Image Operative", Actes d'un séminaire et recueil d'articles, Centre d'Education Permanente, Dept Ergonomie et Ecologie Humaine, Université Paris I.

Rappaport, A.R., & Gaines, B., (1988), "Integration of Acquisition and Performance Systems", *Proceedings of the Integration of Knowledge Acquisition and Performance Systems Workshop, AAAI-88*, St Paul, Minnesota, August 21.

Rumelhart, D.E., Smolensky, P., McClelland, J.L., & Hinton, G.E., (1986), "Schemata and Sequential Thought, Processes in PDP Models", in McClelland, J.L., Rumelhart, D.E., & the PDP Research Group (eds.), *Parallel Distributed Processing*, vol.2, MIT Press, Cambridge, MA, pp. 7-57.

Quillian, M.R., (1968), "Semantic Memory", in M. Minsky (ed.), *Semantic Information Processing*, MIT Press, Cambridge, MA, pp. 216-260.

Rasmussen, J., (1986), *"Information Processing and Human-Machine Interaction: An Approach to Cognitive Engineering"*, North-Holland, New York.

Reason, J., (1986), "Decision Aids: Prothesis or Tools ?", *Intelligent Decision Support in Process Environments*, NATA Workshop, Nov. 11-14, Ispra, Italy.

Rosenschein, S.L., & Kaelbling, L.P., (1986), "The Synthesis of Machine with Provably Epistemic Properties", In J.F. Halpern, Ed., *Proceedings of the 1986 Conference on Theoretical Aspects of Reasoning about Knowledge*, pp. 83-98, Morgan-Kaufmann Publishers, Los Altos, CA.

Rumelhart, D., & al. (Eds.), (1986), *"Parallel Distributed Processing"*, MIT Press, Cambridge, MA.

Schank, R.C., & Abelson, R.P., (1977), *"Scripts, Plans, Goals and Understanding"*, Laurence Erlbaum, Hillsdale, N.J., pp. 1-68.

Woods, W.A., (1975), "What's in a Link: Foundation for Semantic Networks", in Bobrow D.G., Colins A. (eds.), *Representations and Understanding: Studies in Cognitive Science*, Academic Press, New York, pp. 35-84.

Figures

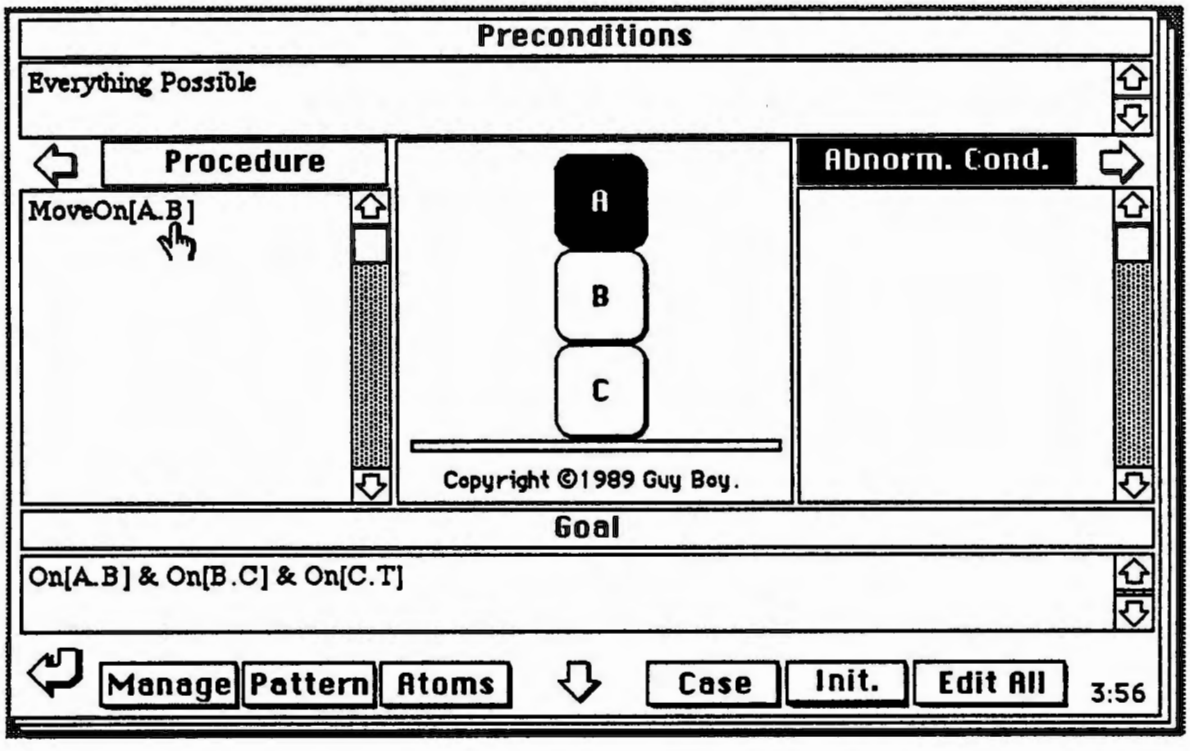


Figure 10. Block of knowledge capable to reach the given goal from the initial situation.

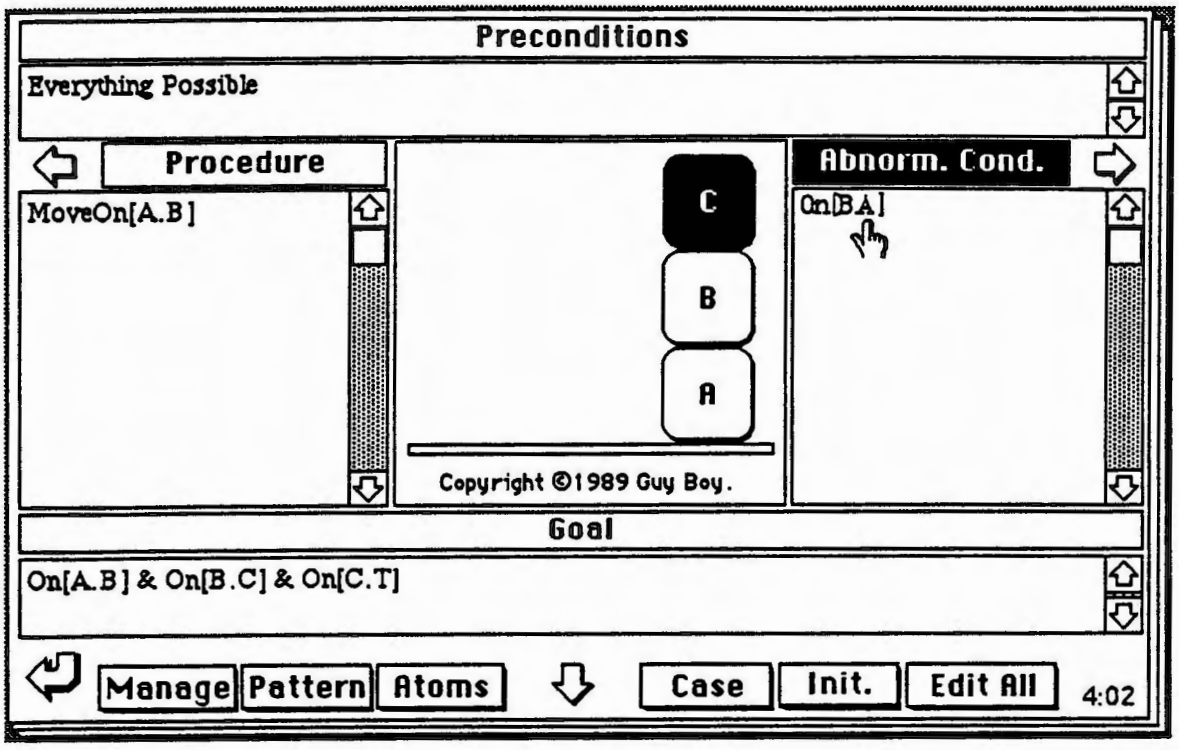


Figure 11. User generating an abnormal condition.

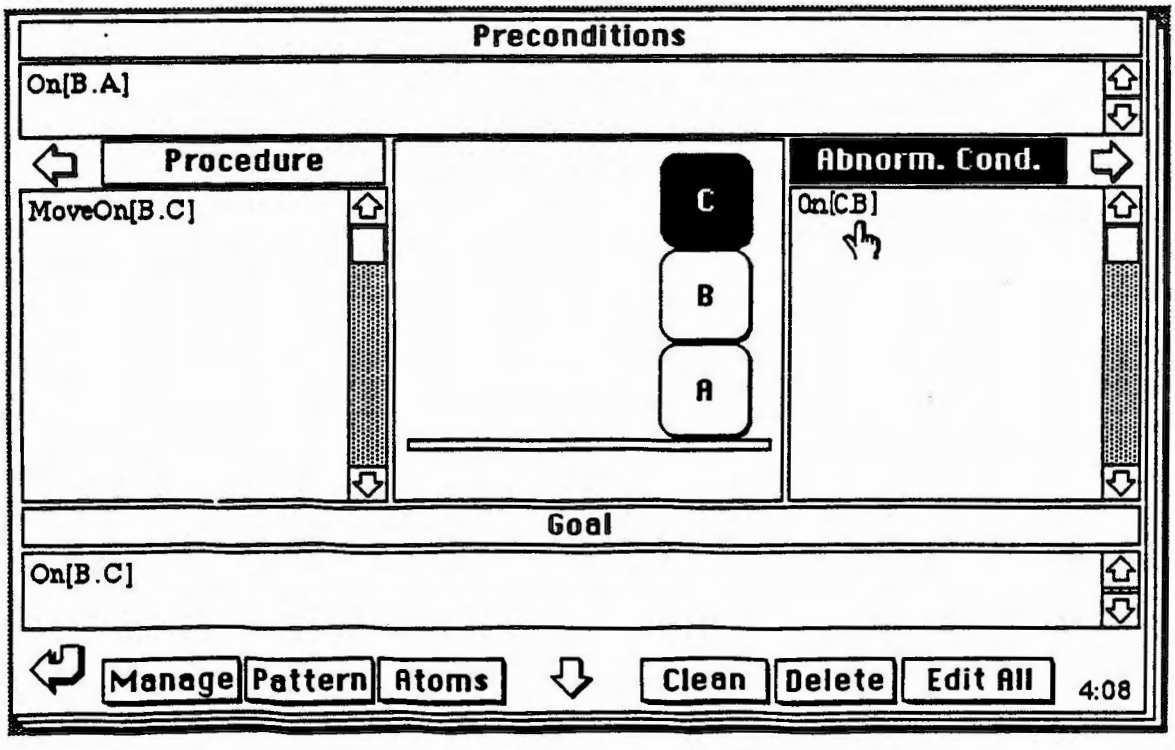


Figure 12. User generating a new abnormal condition.

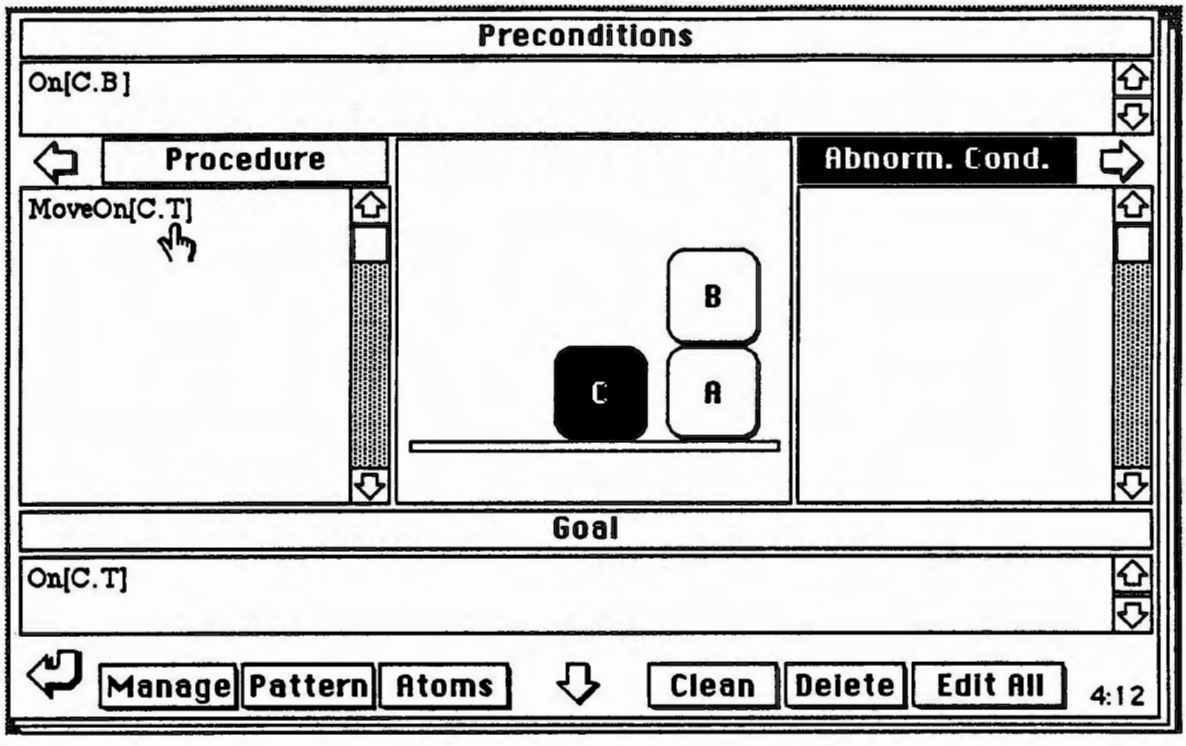


Figure 13. Block of knowledge executing an action to reach its goal.

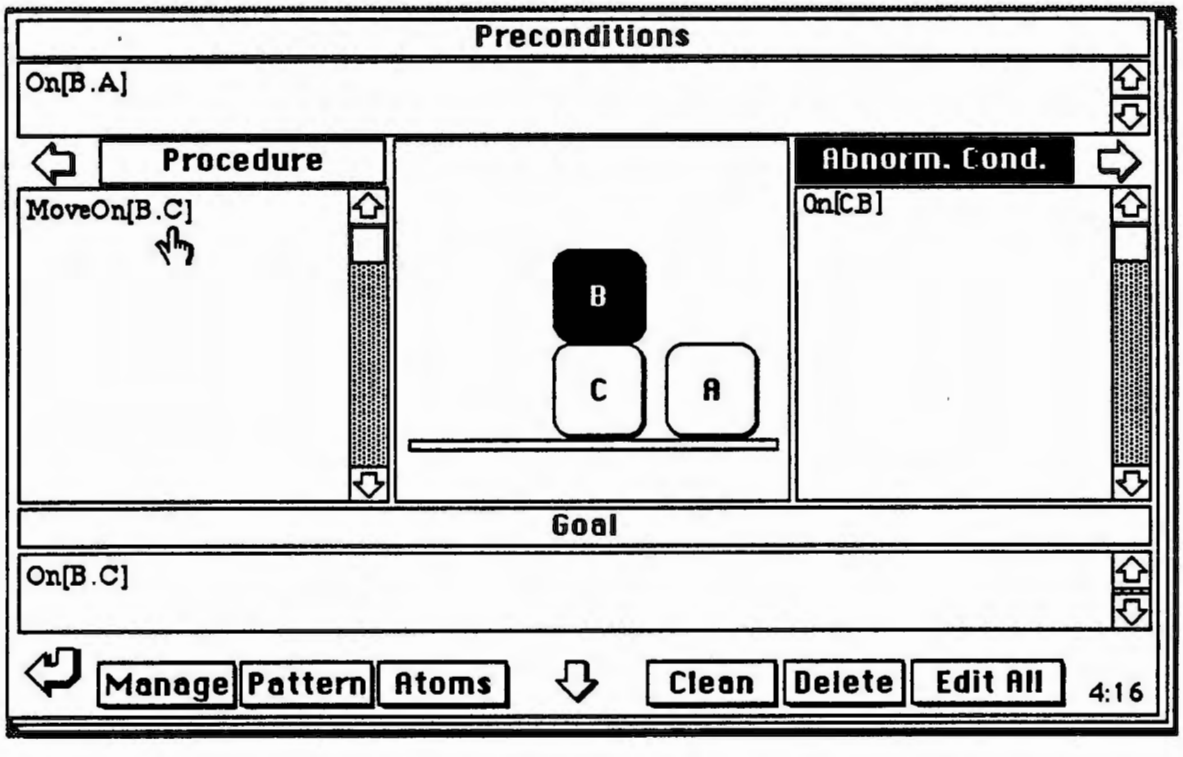


Figure 14. Block of knowledge executing an action to reach its goal.

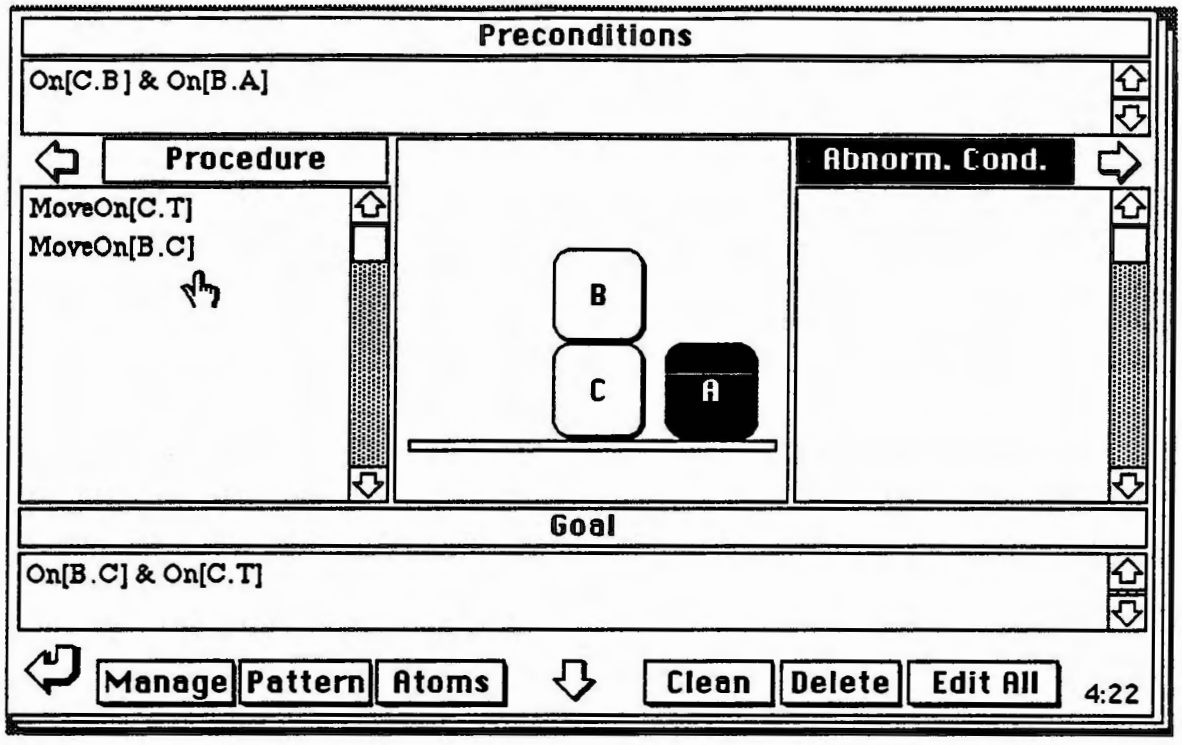


Figure 15. New block of knowledge created by pattern compilation.

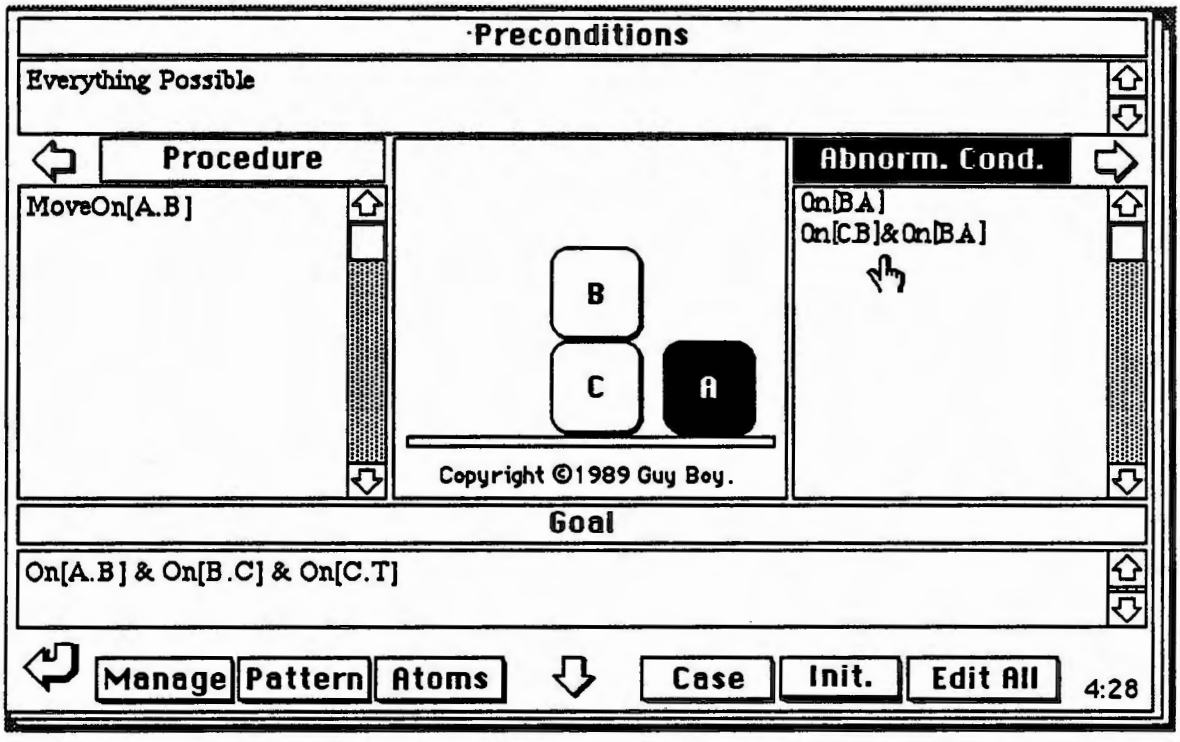


Figure 16. New abnormal condition created by pattern compilation.

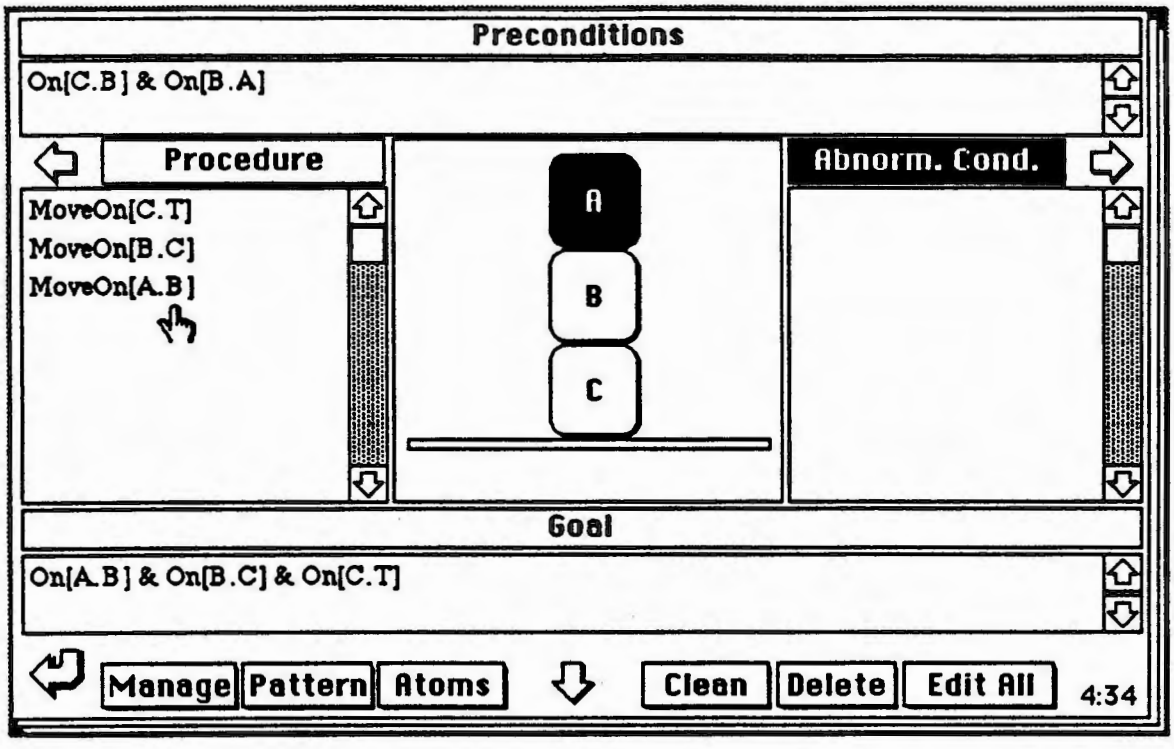


Figure 17. New block (created by pattern compilation) which reaches the overall goal.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989

KWACQ'ing with *DDUCKS:* Knowledge Acquisition Techniques for Decision Analysis Using *Axotl* and *Aquinas*

Jeffrey M. Bradshaw, Stanley P. Covington, Peter J. Russo, and John H. Boose
Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, M/S 7L-64, Seattle, Washington 98124 USA (206) 865-3422; jbrad@atc.boeing.com

ABSTRACT

The effective application of current decision tree and influence diagram software requires a relatively high level of sophistication in the theory and practice of decision analysis. Research on intelligent decision systems aims to lower the cost and amount of training required to use these methods through the use of knowledge-based systems; however, application prototypes implemented to date have required time-consuming and tedious hand-crafting of knowledge bases. This paper describes the development of *DDUCKS*, an "open architecture" problem-modeling environment that integrates components from *Axotl*, a knowledge-based decision analysis workbench, with those of *Aquinas*, a knowledge acquisition workbench based on personal construct theory. The knowledge base tools in *Axotl* can be configured with knowledge to provide guidance and help in formulating, evaluating, and refining decision models represented in influence diagrams. Knowledge acquisition tools in *DDUCKS* will allow the knowledge to be efficiently modeled, more easily maintained, and thoroughly tested.

1. INTRODUCTION

1.1. Progress in Automated Decision Analysis

The most promising approach to dealing with decision complexity in a consistent, general-purpose manner is decision analysis (Howard & Matheson, 1984). In the past few years, researchers and developers have made important theoretical advances and have implemented several successful systems for automated support of the decision analysis process (review by Horvitz, Breese, & Henrion, 1988). Although a thorough discussion is beyond the scope of this paper, we wish to review three of the important developments that have led to the current state of the art. Following this, we will explain why we think that the development of automated knowledge acquisition tools is crucial to the future of efforts to deliver decision analysis methodology to a wider spectrum of decision makers and domains.

Decision tree software. The development of decision tree software (Figure 1) represented an important milestone in automated decision analysis support (Olmsted, 1982; McNamee & Celona, 1987). Through the use of general-purpose commercial tools, decision analysis techniques have become more widely known and used than ever before. At the same time, there are several drawbacks to the use of decision trees as a representation device. For one thing, they grow exponentially with problem size, making them impractical for problems of significant size. Additionally, the tree metaphor for decision problems often leads participants to confuse chronology with the order of probabilistic expansions.

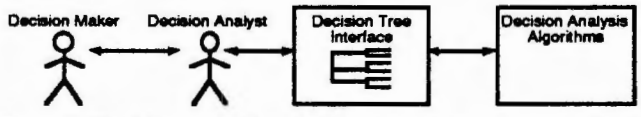


Figure 1. Decision analysis techniques have become more widely used through the availability of decision tree software.

Influence diagram software. Conceived by Howard and Matheson (1981), influence diagrams represent an important advance in the representation of decision problems (Figure 2). While retaining the essentials of the decision theoretic mathematical foundation developed for decision tree manipulation, they provide several advantages over decision trees. Technically, they are superior in that they can explicitly represent and exploit conditional independence relationships between variables. From a practical point of view, they provide a clear and intuitive way of communicating the structure of a decision model. With the advent of commercial software (e.g., Shachter, 1986b) and the assumption of continued success by researchers developing tractable methods for evaluating large influence diagrams (Horvitz, Breese, & Henrion, 1988), we expect influence diagrams to completely supplant decision tree representations.

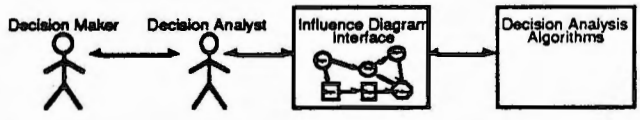


Figure 2. Influence diagrams provide both technical and practical advantages over decision trees.

Intelligent decision systems. Unfortunately, the effective application of both decision tree and influence diagram software requires a relatively high level of sophistication in the theory and practice of decision analysis. These tools contain some of the *algorithms* of decision analysis practice, but cannot embody the *experience* and *intuition* of decision analysis professionals in formulating and appraising decision models. Also, because these tools cannot conveniently store and reuse domain expertise, they cannot exploit the similarity between recurring decisions in the same domain. New decisions are typically modeled from scratch.

The importance of these issues led Holtzman (1989) to define an approach for a third generation of automated decision analysis software called *intelligent decision systems* (IDS)[1].

[1] Related approaches to combining decision analysis and knowledge-based systems have been developed by Breese (1987), Keeney (1986), Moore & Agogino (1987), and Wellman (1986).

An IDS combines a set of automated decision analysis tools with a knowledge-based system that helps decision makers without extensive, treaining in decision analysis build, evaluate, and refine decision models in some well specified domain (Figure 3). To build an IDS application for a class of decision problems, decision analysts and domain experts work with a knowledge engineer to configure the system with "application-independent"[1] knowledge (i.e., knowledge of decision analysis tools and methodology) and application-specific knowledge (i.e., knowledge about a particular domain). Once built, these knowledge bases can be used again and again to provide guidance and help during consultations with decision makers. Task-level consultation interfaces pose questions and interpret results in language and graphical presentations tailored to the decision maker, rather than in terms of standard decision analysis concepts.

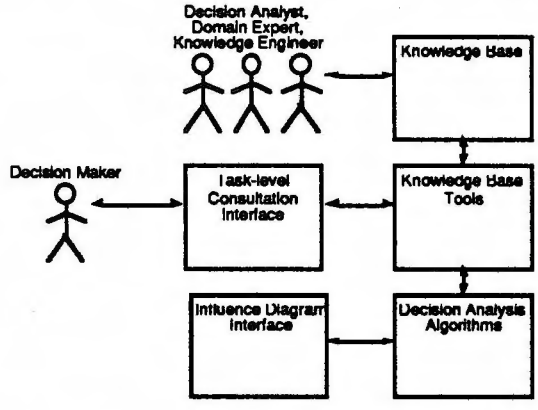


Figure 3. Intelligent decision systems (IDS) combine decision analysis tools with a knowledge-based system

Through implementing a general-purpose IDS architecture *(Axotl)* and applying it to the domain of R&D project selection *(PIE)* we have become convinced that decision analysis can be delivered effectively and economically through the use of knowledge-based systems (Bradshaw & Holtzman, 1987; Bradshaw, Covington, & Russo, 1989). We found that the development of knowledge bases for such a system was relatively straightforward compared to other knowledge-based systems we had created because we were dealing with a methodology (decision analysis) that was mature and rigorously defined. However, we were dissatisfied that so much hand-crafting of knowledge bases had to be performed by knowledge engineers. Through the development of knowledge acquisition tools, we hoped that much more of the knowledge base could be constructed by domain or decision analysis experts themselves.

1.2. Knowledge Acquisition for Fourth-Generation Decision Analysis Support Tools

Figure 4 shows how automated knowledge acquisition tools would fit within an architecture for advanced decision analysis support tools. Knowledge acquisition tools could help cut down the lengthy and error-prone revise-and-review cycle in the development of such systems, making delivery of IDS applications practical on a large scale. While it is unrealistic to expect that the role of knowledge engineers would entirely disappear, their participation in many aspects of knowledge base development and maintenance could be minimized. Figure 4 explicitly includes interfaces to conventional software and external data. The success of future systems in practical applications will depend on whether they can be effectively integrated with other software such as databases, spreadsheets, and hypermedia environments.

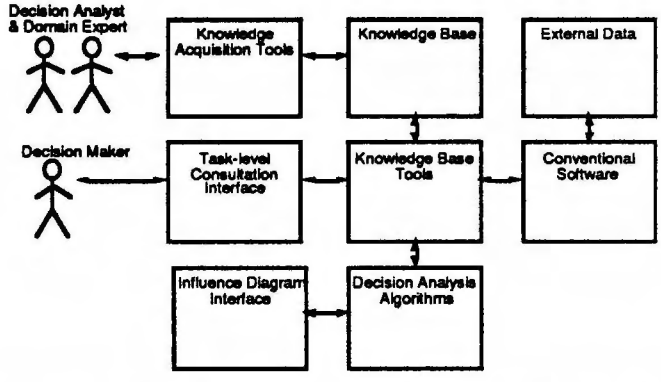


Figure 4. Knowledge acquisition tools and interfaces to conventional software will be an important part of future decision analysis support systems.

Knowledge acquisition research addresses the problem of designing appropriate representations and procedures to facilitate creation, validation, verification, and maintenance of knowledge bases over the lifetime of a knowledge-based system[2]. Thus, it might be said that researchers are attempting to do for knowledge engineering what CASE is doing for traditional software engineering (Bradshaw & Boose, 1989b; Gaines, 1988).

Current work in knowledge acquisition emphasizes that creation of knowledge bases is a constructive modeling process and not simply a matter of "expertise transfer" (Bradshaw & Boose, 1989a). Automated knowledge acquisition tools support the modeling process through the use of role-limiting methods, acquirable knowledge representations, interviewing techniques, and analysis tools.

Role-limiting methods. Many knowledge acquisition tools derive their power from relying on a well defined problem-solving model that establishes and controls the sequences of actions required to perform some task (Gruber, 1988; Klinker, 1989). For example, SALT (Marcus, 1988) is based on a method for design called "propose-and-revise", while

[1] No knowledge is truly application-independent; perhaps "multiple-use" knowledge would be a better term.

[2] Selected papers from the annual European and AAAI-sponsored North American knowledge acquisition workshops are published in the *International Journal of Man-Machine Studies, Knowledge Acquisition: An International Journal*, and in a continuing series of Academic Press books edited by John Boose and Brian Gaines. Boose (1988) presents a survey of current knowledge acquisition techniques and tools.

MOLE (Eshelman, 1988) uses a method of heuristic classification called "cover-and-differentiate". The problem-solving method defines the kind of knowledge applicable within each step, thereby making explicit the different roles knowledge plays. Once these roles are defined, knowledge acquisition tools appropriate to each kind of knowledge are designed.

Acquirable knowledge representations. Knowledge acquisition tools attempt to minimize the problem of representation mismatch, the disparity between a person's task-level description of the problem and its realization in some computable form (Gruber, 1988). This problem is addressed by designing interfaces that bear a strong resemblance in appearance and procedure to the original task (e.g., cancer-therapy protocol forms in OPAL (Musen, 1988); engineering notebooks in vmacs (Sivard, Zweben, Cannon, Lakin & Leifer, 1989)) or by relying on some familiar, high-level, generic knowledge representation metaphor (e.g., repertory grids in ETS, (Boose, 1985, 1986)). Structured high-level representations are used to maintain the knowledge base, but if necessary they can be transformed into other forms such as rules. Transformation of knowledge into multiple forms for visualization purposes is useful as a means of facilitating insight.

Interviewing techniques. A number of interviewing techniques have been developed or borrowed from fields such as psychology in order to guide experts through the knowledge acquisition process. For example, some of the techniques originally developed by Kelly (1955) to discover the conceptual structure of clients in psychotherapy have been applied to knowledge acquisition.

Analysis tools. Both formal and heuristic modes of analysis are available within many knowledge acquisition tools as a means of verifying and refining the knowledge base. Shaw and Gaines (1988), for instance, have developed a methodology for analyzing areas of consensus, conflict, correspondence, and contrast between the conceptual systems of two or more experts. Many knowledge acquisition tools have also been integrated with consultation systems, making sophisticated performance testing of the system possible (e.g., Boose, Shema & Bradshaw, 1988).

1.3. *DDUCKS*: An Integrated Environment for Knowledge Acquisition and Decision Analysis

Recently, we have begun development of an environment called *DDUCKS* (Decision and Design Utilities for Comprehensive Knowledge Support)[1]. The environment integrates components from *Aquinas*, a knowledge acquisition workbench based on personal construct theory (Boose & Bradshaw, 1987; Boose, Shema & Bradshaw, 1988; Boose, Bradshaw, Kitto, & Shema, 1989), with those of *Axotl*, a knowledge-based decision analysis workbench (Bradshaw, Covington, and Russo, 1989). The "open architecture" design of the *DDUCKS* environment allows knowledge-based coordination of local or networked applications such as spreadsheets, databases, or hypermedia software. One of *DDUCK's* components, MANIAC (MANager for InterApplication Communication) supports asynchronous and synchronous communication between any number of multitasking applications. *DDUCKS* runs on Apple Macintosh II hardware; subsets of *DDUCKS* run in other environments.

In the remainder of this paper, we will describe the decision analysis workbench and knowledge base tools of *Axotl*. Then we will outline the knowledge acquisition tools being developed to support the creation of content and process knowledge bases for applications of *Axotl*.

2. APPROACH

2.1. *Axotl:* A Knowledge-Based Decision Analysis Workbench

Axotl combines a general-purpose decision analysis workbench with knowledge base tools to assist individuals consulting with the system about a specific problem. The first version of *Axotl* was developed as part of a joint effort by Boeing Computer Services and Strategic Decisions Group (SDG) to build a flexible, general-purpose intelligent decision system that could be applied to R&D management decisions. The application is called PIE, for Project Investment Evaluator. Boeing Computer Services, with its experience in building knowledge-based systems (Bradshaw & Boose, 1989c), contributed expertise in software engineering, knowledge engineering, and Boeing-specific knowledge on this project. SDG has a major practice in applying decision analysis to R&D management decisions. In addition, SDG staff have pioneered the theory and practice of decision analysis (Howard & Matheson, 1984) and had previously implemented intelligent decision system application prototypes (Holtzman, 1989).

Following the successful demonstration of knowledge-based guidance of a decision analysis consultation in mid-1988, it was decided that Boeing and SDG would pursue further development work on the system independently. Since that time, we have made a number of extensions to *Axotl*, and have developed *DDUCKS* as the integrating framework for linking *Axotl* to *Aquinas* and other applications. We have loaned *DDUCKS* to a medical non-profit organization for evaluation of its applicability to bone marrow transplant follow-up care (Sullivan & Shulman, 1989). We are also applying portions of the environment to facilitate demonstrations of design knowledge capture for a NASA-sponsored Corporate Memory Facility project (Bradshaw, Boose, Covington & Russo, 1989; Boose & Bradshaw, 1989; Bradshaw & Boose, 1989d), and in business process management in a Boeing quality improvement context (Bradshaw, Covington, & Kipersztok, 1989). SDG has also enhanced their version of the software, and is developing a commercial application to R&D management, called *R&D Analyst*™[2]. Holtzman and Seiver (1988) have developed a

[1] Either the first or second *D* in *DDUCKS* is silent, depending on whether one using the tool in a decision or design context.

[2] *R&D Analyst* is a trademark of Strategic Decisions Group. All rights reserved.

successful application of the system that assists critical care clinicians in ventilator management decision-making.

Axotl was developed on the Apple Macintosh II (*MacXotl*) and runs on all platforms that support ParcPlace Smalltalk-80 (e.g., Sun, Apollo, Hewlett-Packard, IBM '386 compatibles). We will first describe the decision analysis workbench, then the knowledge base tools.

2.1.1. Decision Analysis Workbench

The decision analysis workbench contains a graphical editor that is used for creating and refining models of alternatives, preferences, and uncertainties relevant to a specific decision. These models, called *influence diagrams*, are solved to obtain recommended actions in a way that is consistent with probability and utility theory (Howard & Matheson, 1981).

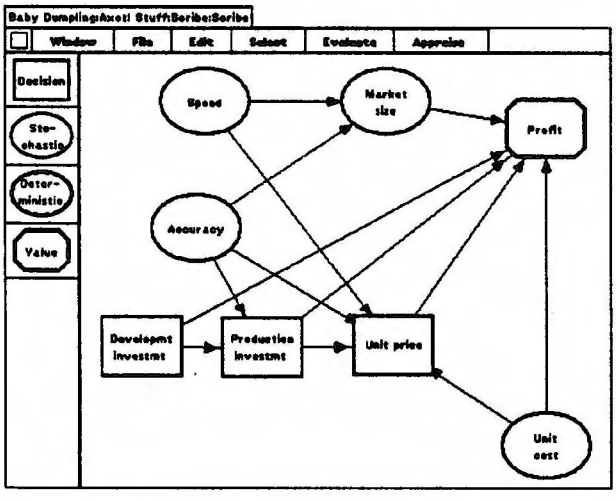


Figure 5. An influence diagram for an R&D investment decision.

Figure 5 shows a screen snapshot of an *Axotl* influence diagram (Howard & Matheson, 1981) representing a prototype R&D project decision problem. The problem is to determine an investment strategy for "Scribe" an automated speech-to-text transcriber, taking technical risks and market uncertainties into account. The investment strategy is composed of three decisions (development investment level, production investment level, and unit price), represented by rectangular nodes on the diagram. Oval nodes represent technical risk variables (accuracy, speed), production uncertainties (unit cost), and market uncertainties (market size). The eight-sided node labeled "Profit" has been designated as the criterion to maximize in evaluating the decision model to determine an optimal policy. Arrows between nodes represent relationships where influence or information is imparted from one variable to another. For example, arrows from "Speed" and "Accuracy" to "Market size" represent the assertion that the degree of technical achievement in these areas will affect the size of the market for "Scribe". The arrows from "Speed" and "Accuracy" to "Unit price" indicate that these uncertainties will be known at the time a pricing decision is made. An additional type of node, not shown in this diagram, can represent a deterministic function. This facilitates transparent links from the influence diagram to external procedures and functions or to programs such as spreadsheets.

The method of solving influence diagrams implemented in *Axotl* incorporates a new approach developed by Smith (1988) that allows a wide range of questions to be answered directly from the diagram. It preserves the entire underlying joint distribution during solution and inference procedures, rather than just the value lottery and decision policy as is usually done (e.g., Shachter, 1986a). The distribution editor, used to structure conditional probability distributions, is another feature unique to *Axotl*. It introduces the concepts of *coalescence* (i.e., sharing of atomic distributions) and *clipping* (i.e., explicit pruning of impossible or unnecessary conditioning paths and their atomic distributions).

One of the most significant results of *Axotl* development was the formulation of generic procedures for the use of influence diagrams in valuing information and control (J. Matheson (1988) discusses some of these issues). An approach was developed to allow the automated conversion of any influence diagram to "Howard Canonical Form" so that value-of-perfect-information or -imperfect-information questions could be answered for any variable. Based on an understanding that value-of-control calculations can readily be interpreted only for causal influence diagrams, we developed heuristic techniques that specified when it was appropriate to ask value-of-perfect-control or -imperfect-control questions and to formulate generic procedures for answering them. Eventually, these techniques could be fully automated.

2.1.2. Knowledge Base Tools

When configured with the appropriate knowledge, the knowledge base tools in *Axotl* guide decision makers through the process of formulating, evaluating, and refining a decision model in a specific domain. The model is kept to tractable size by deliberately limiting the problem domain for a given application and using heuristic methods represented in knowledge bases to guide influence diagram construction. These heuristics can be regarded as instances of the types of categorical methods described by Szolovitz and Pauker (1978):

> "When the complex problems need to be addressed—which treatment should be selected, how much of the drug should be given, etc.—then causal or probabilistic models are necessary. The essential key to their correct use is that they must be applied in a limited problem domain where their assumptions can be accepted with confidence. Thus, it is the role of categorical methods to discover what the central problem is and to limit it as strongly as possible; only then are probabilistic techniques appropriate for its solution."

We distinguish between two major types of knowledge in the knowledge base: content and process.

Content knowledge. Content knowledge is substantive; it is the *what* of decision model building during a consultation. Internally, content knowledge is represented as a set of

partially defined influence diagram variables for a class of decisions, their interrelationships, and the conditions under which they may be added to or removed from the influence diagram being constructed.

During the knowledge acquisition phase of the PIE project, we constructed a "big influence diagram" showing all of the structure that would be considered during the consultation process. Once agreement was reached about this structure, it was further broken down into smaller, overlapping groups of related variables along with some possible value models. We found that the "big influence diagram" for R&D project decisions had three distinct stages (research, development, application) and three possible value models (contract, market, simple value).

These value models and groups of influence diagram variables serve as the building blocks for the first-cut decision model that is constructed during subsequent consultations with a decision maker. Through a process called *template development*, the system assk the decision maker questions about the project that will help it determine the stage of development and the appropriate value model (D. Matheson, 1988). The system uses this project information to construct a generic, skeletal template. The dialogue continues to help decision makers expand and refine the initial model according to their specific circumstances.

Process knowledge. Process knowledge is strategic; it is the *how* of decision model building during a consultation. It consists of generic and domain-specific expertise about what to do at each stage of the decision-making process. Internally, process knowledge is represented by activity graphs and rules.

Figure 6 represents a view of the process of decision making for R&D projects that we used in developing the PIE application. Project investment decision making is typically done at two levels: at the portfolio level and at the level of the individual project. In PIE, we focused only on project-level decisions.

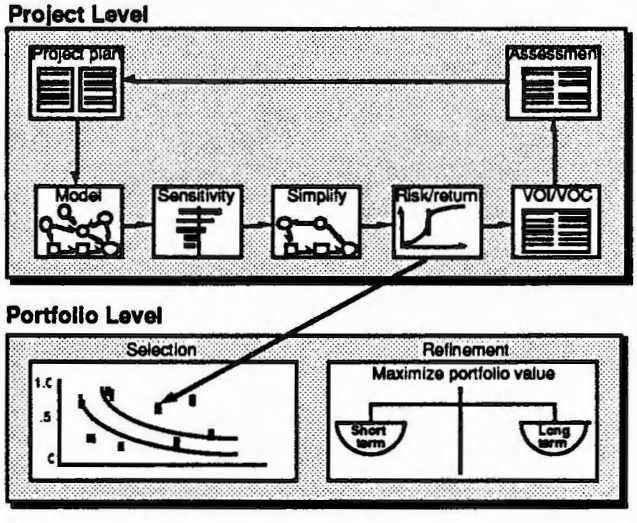


Figure 6. An overview of project investment decision making.

The cycle shown in Figure 6 represents general process knowledge about what to do at each stage of the R&D decision making process. We used the detailed process knowledge we acquired from decision analysis and Boeing experts to build a prototype R&D project selection decision activity graph (Figure 7) and rule base. Consultations using this knowledge base proved the ability of PIE to pose a series of questions to a Boeing manager about a project under consideration and to formulate, evaluate, and appraise an appropriate influence diagram model based on answers to these questions. The decision analysis tools ran under control of the knowledge base, and the appropriate decision analysis task modules were invoked by the knowledge base at different times to evaluate the model and provide answers to value-of-information and value-of-control questions. The influence diagram was linked to a Microsoft Excel spreadsheet model containing financial formulas. Facilities in *Axotl* allowed Excel to be launched and the spreadsheet computations made automatically during influence diagram solution.

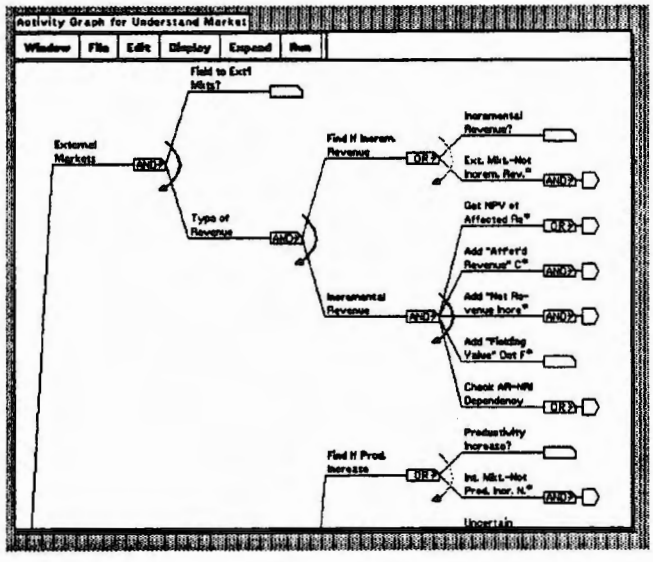


Figure 7. A portion of the R&D project selection activity graph.

An activity graph is a representation of the consultation process as a hierarchy of goals and activities (Holtzman & Russo, 1988). There are two kinds of nodes: goals and activities. The topmost goal in the hierarchy represents the successful completion of a consultation; any number of subgoals may be added. At the leaves of the graph, activities represent procedures that may be executed in support of goals during the course of a consultation (e.g., "assess probability of technical success," "calculate value-of-information on market size"). Successful completion of a consultation requires that a sufficient set of goals be satisfied, either through the execution of the supporting activities or through being explicitly declared satisfied by the individual.

The activity graph facility includes components for viewing and editing goals and activities graphically. Activity graphs can call other activity graphs. During a consultation, an agenda is constructed from a "cut set" through the activity graph. The activities are executed one by one until the failure

of an activity or some other change in conditions necessitates a modification of the agenda. The activity graph and agenda are invisible to the decision maker, who sees only the consultation interface.

A knowledge base tool called the heuristic advisor selects and modifies activity graphs during consultations. The knowledge base for the heuristic advisor is currently represented as rules and facts within an MRS-like inference engine (Russell, 1985), implemented in Smalltalk-80.

2.2. Knowledge Acquisition Tools for *Axotl*

Knowledge acquisition tools are being implemented in *DDUCKS* to reduce development time and effort and enhance the quality of knowledge bases. These tools are organized around three different roles for content knowledge and two for process knowledge. Each role has one or more forms of representation associated with it; some have interviewing techniques or analysis tools as well.

2.2.1. Content Knowledge

Elements of the decision basis can be separated into three major components: alternatives, information, and preferences (Figure 8).

Alternatives. *DDUCKS* includes*Canard*[1], a knowledge acquisition tool based on possibility tables that can be used to generate and structure complex alternatives (Shema, Bradshaw, Boose, & Covington, 1989). Links are maintained between the tables and decision nodes that are part of the content knowledge base. The possibility table representation is based on the manually developed strategy tables (McNamee & Celona, 1987) and morphological charts (Zwicky, 1969) that have been used by decision analysts and designers for many years. *Canard* automates this representation and extends its logic and structure to allow knowledge-based inference and the representation of more complex problems (e.g., hierarchical tables, explicit representation of constraints).

At one point, we used a manually developed possibility table similar to the one in Figure 9 to help us define alternatives for *Axotl* system development. Alternatives are shown in the leftmost column. Other columns represent important components of the system and various options for development within each one. The path of outlined boxes traced through the columns defines the "base case" alternative. Time and budget requirements and constraints could also be associated with variables in the table. Within *Canard* these constraints are used to "gray out" infeasible options during the definition of a path.

[1] A canard is an airplane that sports a tail in front rather than in back—a symbol of our "backward" analytical approach to synthesis.

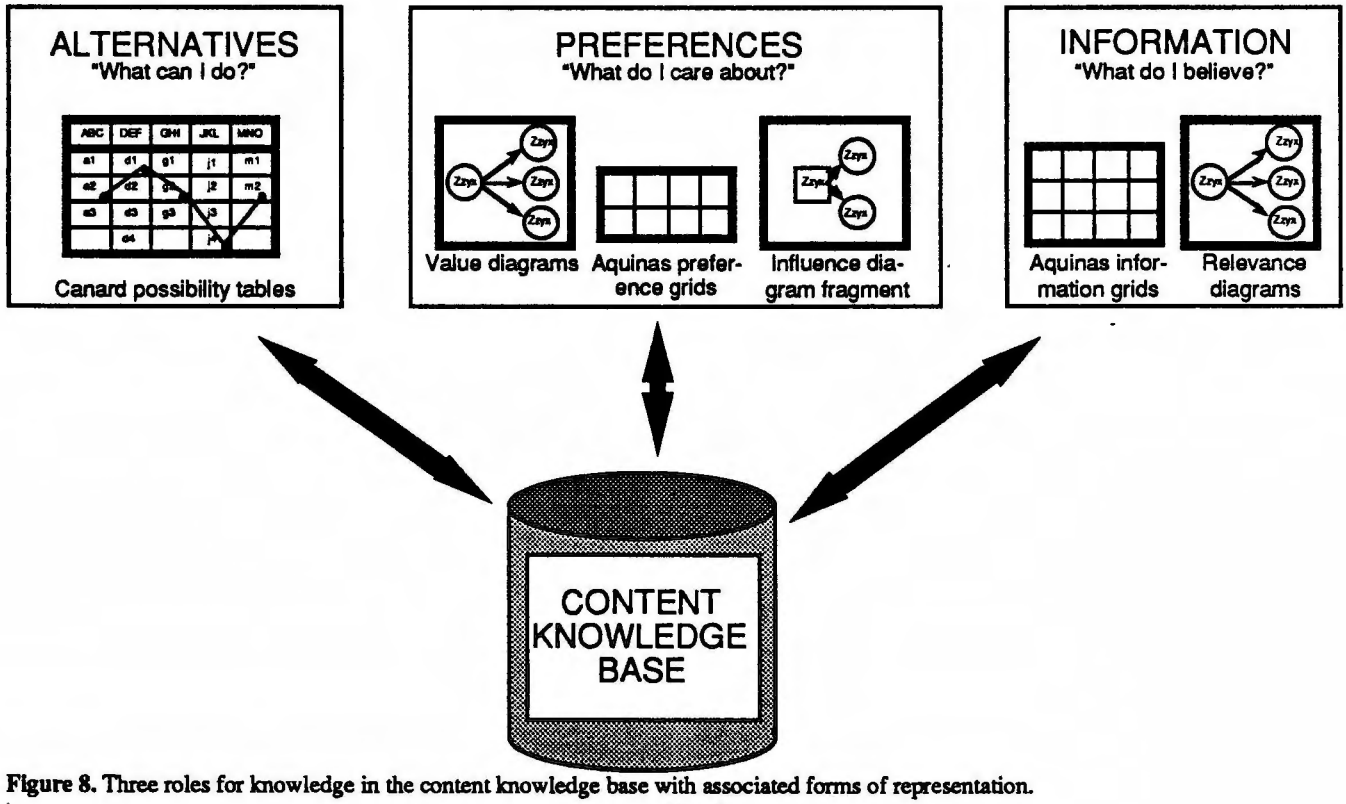


Figure 8. Three roles for knowledge in the content knowledge base with associated forms of representation.

POSSIBILITIES	COMPONENTS								
	Spreadsheet	Home-Brew Prolog	Document'n of Code	Probability Encoder	DA Knowledge Base	Journal and Transcript	Dialog Boxes	UNDO Facility	Tutorial and User Manual
Tight Resources	None 0	Use XQP only 0	No add'l document'n 0	Leave as is 0	None 0	None 0	Sequential menus only 0	None 0	None 0
Base Case 2	Modify Modify existing spreadsheet 5	Straight port of IE w/o XQP emulation 2	Cursory update of PD document 3	Complete wheels and bar charts 1	Single, skeletal KB 5	Journal only 1	Ad hoc dialog boxes (no generality) 2	Support only a few commands 1	Terse, thorough primer 3
Flashy Demo	Enhance existing spreadsheet 9	IE with XQP syntax emulation 3	Throrough update of PD document 5	Add cumulative display 2	Single, saleable KB 12	Basic transcript only 3	Ad hoc but flexible dialog boxes 3	Support most commands 3	Verbose primer 5
All-out	Include some fancy features 5	IE with XQP & optimization 5		Add discretization 4	Two saleable KB's 15		Generic dialog box facility 7	Multiple UNDO's up to a definable threshold 5	Full-fledged manual with tutorial 10
	Comm'l qual spreadsheet 40				Fancy, useable KB 35	Journal and fancy transcript 5			

Figure 9. A possibility table defining *Axotl* development alternatives.

A major concern is helping people better explore the space of alternatives. Cognitive scientists have long known that humans typically retrieve only a small fraction of available alternatives when generating hypotheses. People tend to anchor on initial guesses, giving insufficient regard to subsequent data. For a variety of reasons, people may not be able to visualize whole classes of possibilities. Although it would be impossible in practice to guarantee that all relevant alternatives were indeed identified, *Canard* can help people consider a richer set of alternatives. An iterative search procedure that proposes new alternatives based on permutations of the constraint space assists in generation of alternatives. Through an analogous procedure, the system can hypothesize new constraints based on examples of previously defined alternatives.

Preferences. Knowledge acquisition tools can be used to help people determine important dimensions of value associated with the alternatives. Within *Aquinas*, a number of useful representation, interviewing, and analysis techniques have been developed. Many of these techniques are based on the research of George Kelly (1955), a clinical psychologist who emphasized the foundational role of distinctions (*personal constructs*) underlying the processes of perception and reasoning. For example, using a *triadic elicitation* interviewing technique, *Aquinas* would ask people to define preferences by considering groups of alternatives presented three at a time: "Think of an important consequence that two of A1, A2, and A3 share, but that the other does not. What is that characteristic?" After giving B as that characteristic, the person would be asked about A2, A3, and A4 and come up with a second characteristic C, and so forth.

One way of representing this information is through a *repertory grid* (Figure 10). A grid is a matrix with elements (i.e., alternatives or outcomes) ranged along the bottom and constructs (i.e., dimensions of similarity and difference between elements), defined by extension, as a horizontal row of values (or probability distributions) within the matrix. Although the grid and network representations in Figure 10 are logically equivalent, we find both of them useful and complementary as problem clarifiers from a human factors point of view (Jones, 1981). The grid presentation allows the person to see patterns of similarity and difference that would otherwise be difficult to grasp. Analysis techniques in *Aquinas* (e.g., similarity analysis, cluster analysis) exploit these patterns to help users discriminate more carefully among similar concepts as part of model refinement. Implication analysis helps users discover important dependencies between constructs. A more complete discussion of the relationship of personal construct and decision analytic methods is given in Bradshaw and Boose (1989a).

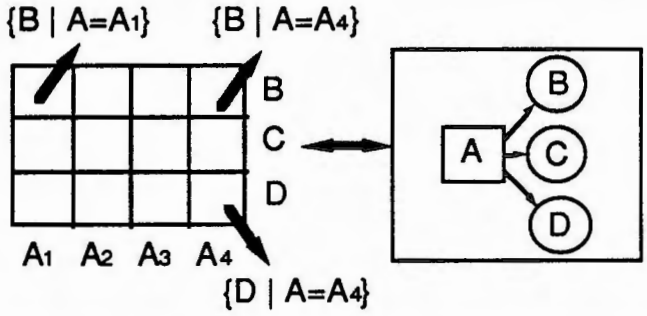


Figure 10. A preference grid and its corresponding influence diagram representation.

Information. Many of the same representation, interviewing, and analysis techniques mentioned above can be used to develop information models for the knowledge base.[1] Interviewing techniques (e.g., laddering) develop the structure of the networks by a recursive expansion procedure that terminates when all leaf nodes are directly observable. Heckerman has developed a related method called *similarity networks*, which uses techniques similar to those used in personal construct methodology to identify and display relationships indicating constraints on conditional independence relationships (Horvitz, Breese, & Henrion, 1988). Figures 11 and 12 show equivalent information grid and relevance diagram representations. When the structure of the dependencies between variables is simple, the transformation between grids and influence diagrams is

straightforward. For complex dependencies, some of the methods described by Howard (1988) in his paper on *knowledge maps* may be useful. These methods allow complex probabilistic assessments to be broken down into a series of simpler ones by manipulating the conditioning relationships between chance nodes. We are sponsoring a project with Seattle University to develop procedures to make the use of disjoint and redundant knowledge maps practical in *Axotl*.

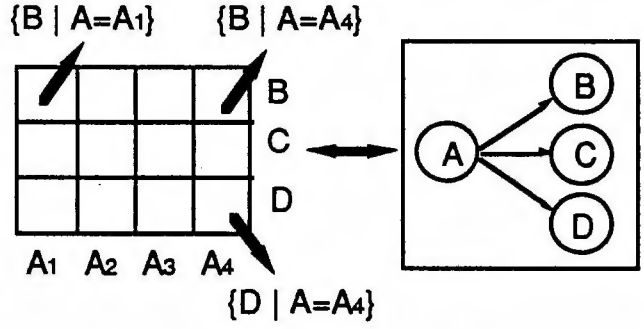


Figure 11. A simple information grid and its corresponding influence diagram representation.

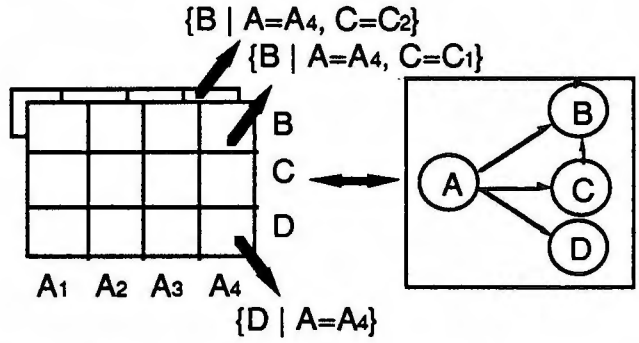


Figure 12. A more complex information grid and its corresponding influence diagram representation.

Figure 13 illustrates some of the relationships possible between elements in the content knowledge base and the structure of a particular influence diagram. Note that the direction of conditioning between chance nodes during the knowledge acquisition process is often the reverse of the direction that is required when the influence diagram is constructed.

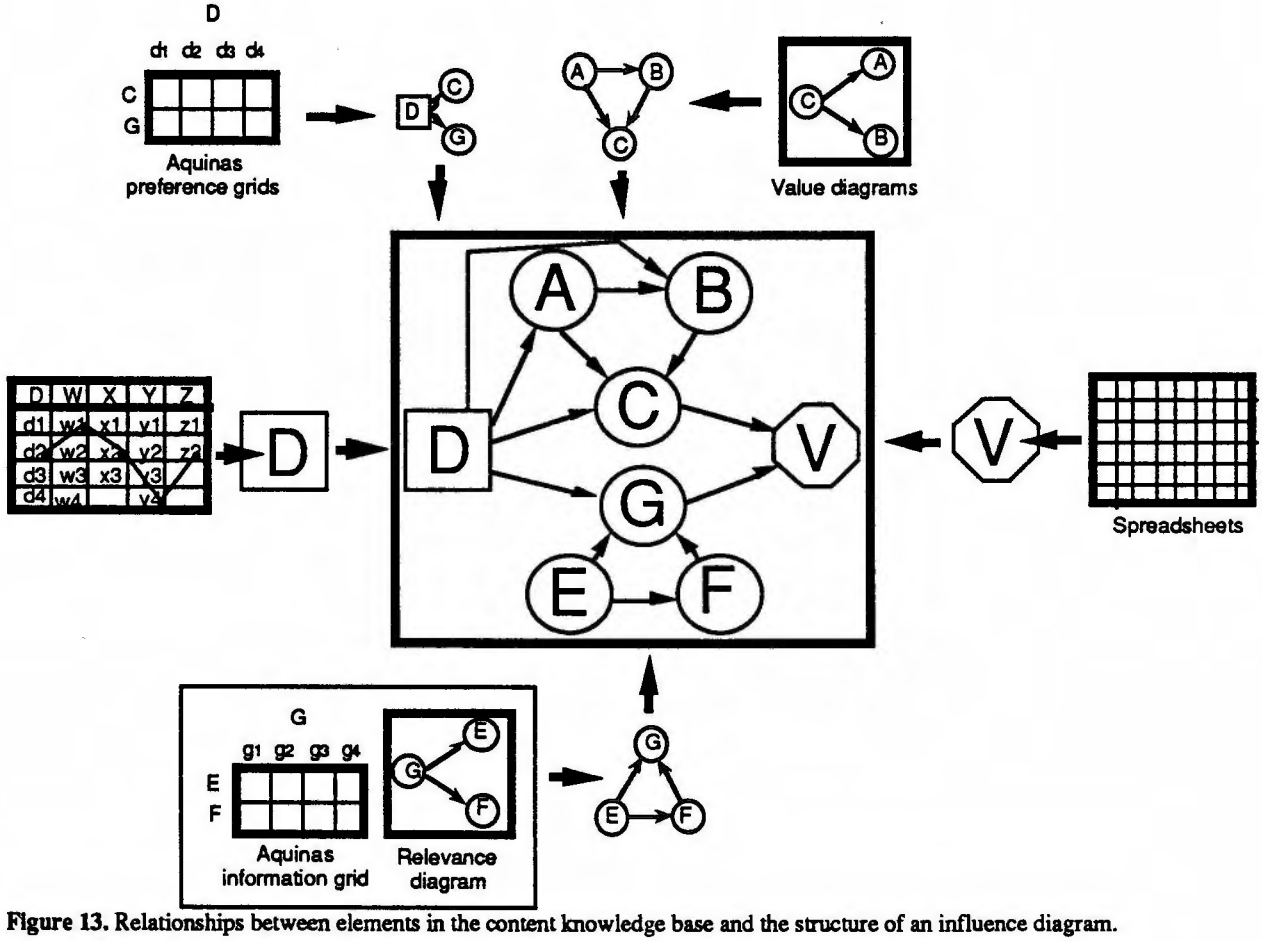


Figure 13. Relationships between elements in the content knowledge base and the structure of an influence diagram.

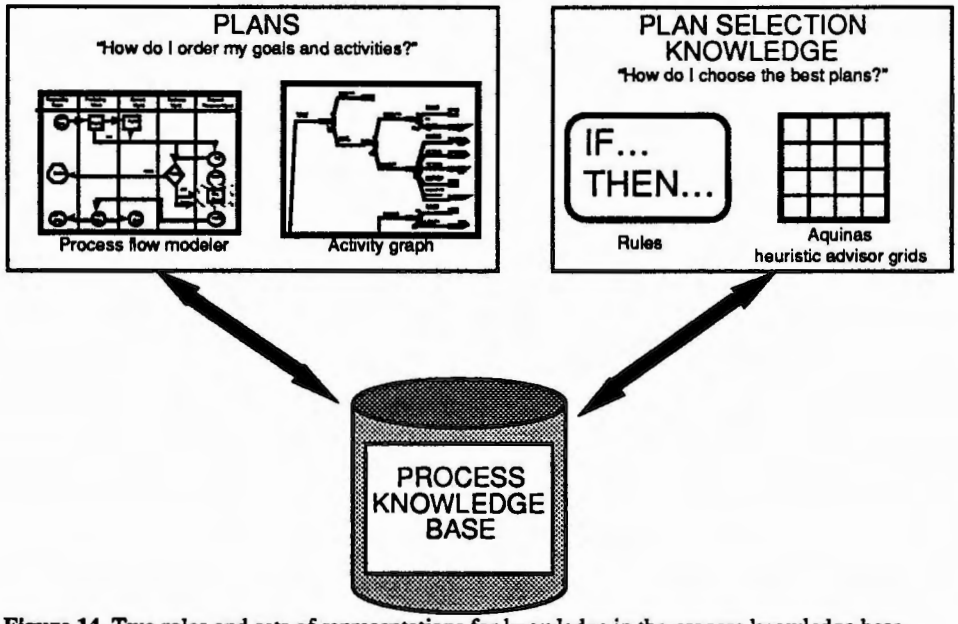


Figure 14. Two roles and sets of representations for knowledge in the process knowledge base.

2.2.2. Process Knowledge

We distinguish between two major types of process knowledge: plans and plan selection knowledge (Figure 14).

Plans. Despite many advantages as a representational device, activity graphs have proved to be unworkable as a knowledge acquisition interface. For this reason, we are building an alternative means of activity graph entry based on hierarchical process flow charts (Bradshaw, Covington, & Kipersztok, 1989). Tasks described as process flows are automatically transformed into executable activity graphs (Figure 15). During a consultation, this activity graph is further compiled into an agenda, as described above[1]. By associating time and resource estimates with the activities, analyses and simulations can be performed that reveal bottlenecks or other constraints in the process.

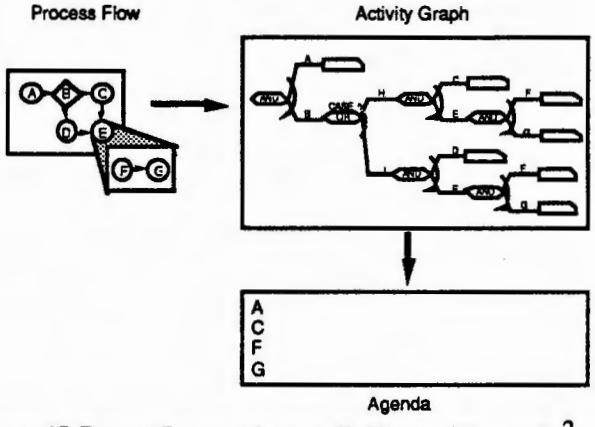


Figure 15. Process flows can be compiled into activity graphs[2].

Plan selection knowledge. Recently, we have begun to think about what it would take to create good knowledge acquisition tools for this type of knowledge. We have started by creating graphical editors for world trees, rules, queries, and attributes that have simplified the construction of rules in knowledge bases. In the future, links to *Aquinas* will facilitate the construction of knowledge bases for the heuristic advisor. The heuristic advisor is a knowledge base tool that evaluates activities with respect to criteria such as completeness, balance, precision, and cost. Based on this evaluation, the heuristic advisor modifies the activity graph to re-order activities dynamically during a consultation. A related effort has dealt specifically with reasoning about computational resource tradeoffs (Horvitz, 1987). In contrast to our heuristic approach, Horvitz has developed these ideas within a formal framework.

3. DISCUSSION

The heart of our approach to making knowledge acquisition simpler for intelligent decision systems is "divide and conquer". Having experienced the development of many knowledge-based system prototypes using automated knowledge acquisition tools, we are convinced that such a strategy will permit experts to do much of their own knowledge engineering, without requiring a great deal of specialized training (Boose, Bradshaw, Kitto, & Shema, 1989). To the experienced decision analyst or knowledge engineer, the development of these tools may seem completely unnecessary; there is little question that such a person would find it more efficient to work directly with influence diagrams and activity graphs than with the knowledge acquisition tools we are defining. In addition, one

[1] The tools can be used for general process management in addition to controlling a consultation.

[2] This figure is somewhat simplified for illustrative purposes. In reality, a separate activity graph would be compiled for the main process flow and the subprocess (F->G). This subgraph would be called by the main graph dynamically during execution.

could argue that by making it too easy on the naive users of a system, putting everything in their terms without requiring them to come to grips directly with the underlying methodology, we are promoting a "black box" mentality that makes it impossible for them to step in when the system breaks down or encounters a problem it can't solve.

There is no easy answer to these objections; there seems to be a real and inevitable tradeoff between the "acquirability" and the expressive power of knowledge representations (Gruber, 1988). Figure 16 shows this tradeoff as a dark curved line. The most powerful means of getting an idea into a computer is programming; unfortunately, even with advances in software engineering, programming remains the most difficult form for nonspecialists to express their knowledge in. On the other hand, form-filling interfaces that correspond to the way a user normally enters information on paper are very easy to learn but are very rigid and limit the applicability of the tools to the very specific problems that the system designer has foreseen. Knowledge acquisition tools do not eliminate the competition between acquirability and expressive power, but they can act as a kind of magnet to help pull the curve out (light gray line). Through the application of automated techniques, acquirable interfaces can become more powerful, and powerful interfaces can be more easily learned and used. *Aquinas* and *Axotl* make the power of influence diagrams accessible to a wider range of people.

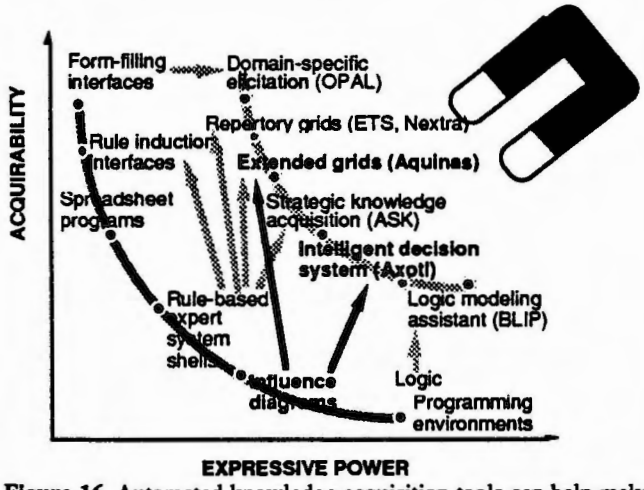


Figure 16. Automated knowledge acquisition tools can help make acquirable interfaces more powerful, and powerful interfaces more easily learned and used (figure adapted from Gruber, 1988).

Because new users will always prefer acquirable interfaces and experienced users will prefer powerful ones, the best strategy to adopt when adding new layers of representation between the new user and the underlying problem-solving model is to leave the hooks to the lower level representations intact. Designers of such systems should also find ways to encourage the less experienced users to keep learning and move on to more expert modes of interaction as they become more familiar and comfortable with the system. Experienced users, on the other hand, should be able to go directly to the heart of the system without interference from a "friendly" interface. This is the approach taken in the nested "Russian doll" interface of HyperCard, where users can graduate from browsing to painting to authoring to scripting at their own rate while performing useful work with the system at each level. This is also the consistent with the "glass box" idea advocated by Wenger and Brown at IRL: to design software that promotes understanding of and access to the inner workings of the system (Feinstein, 1989).

Our conjecture is that high-level representations and interviewing and analysis tools will sometimes be helpful to experienced as well as naive users. This is a question that we hope to address as the tools are evaluated in real-world domains.

We are cautiously optimistic about the future of *DDUCKS*. Although current and future applications will no doubt continue to bring new challenges and difficult problems to solve, the tools embody many years of experience in decision analysis automation and knowledge-based system development. Furthermore, the system derives power from its integration of elements that we feel will be the building blocks of future systems for automated support of complex decisions: a decision analysis workbench for sound reasoning under uncertainty and resolution of preference issues; a knowledge-based system for provision of help in building formal decision models and to promote the re-use of domain knowledge; and a collection of knowledge acquisition tools tailored to the domain that allow such systems to be efficiently modeled, more easily maintained, and thoroughly tested.

ACKNOWLEDGEMENTS

We express our appreciation to Kit Bakke, Miroslav Benda, Kathleen Bradshaw, Beverly Clark, Karolyn Covington, Mike Freeman, Peter Friedland, Oscar and Sharon Kipersztok, Cathy Kitto, Sandra Marcus, Art Nagai, Guy Olney, Doug Schuler, Kish Sharma, David Shema, Greg Swietek, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. Thanks also to reviewers who provided many helpful suggestions. Samuel Holtzman, Ronald Howard, Keh Shiou Leu, James Matheson, Léonard Bertrand, and James Smith of Strategic Decisions Group were co-designers and co-developers of the original *Axotl* prototype. This work has also benefitted from ongoing discussions with Brian Gaines, Max Henrion, and Mildred Shaw. *Aquinas*, *Axotl*, *Canard*, and *DDUCKS* are under development at the Boeing Advanced Technology Center. The original development of *Canard* was supported by NASA contract NAS2-12108.

REFERENCES

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

BOOSE, J.H. (1988). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition: An International Journal*, in 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.

BOOSE, J.H. & BRADSHAW, J.M. (1989). Information to be retained in a Corporate Memory Facility. Boeing Computer Services Advanced Technology Center Technical Report BCS-CMF-TR1. NASA Contract No. NAS2-12108, 31 March.

BOOSE, J. H., BRADSHAW, J.M., KITTO, C.M., & SHEMA, D.B. (1989). From ETS to Aquinas: Six years of knowledge acquisition tool development. *Proceedings of the Third Annual European Knowledge Acquisition Workshop*, Paris, France, July.

BOOSE, J.H., SHEMA, D.S., & BRADSHAW, J.M. (1988). Recent progress in Aquinas: A knowledge acquisition workbench. *Proceedings of the Third AAAI Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Banff, Canada, November.

BRADSHAW, J. M. & BOOSE, J.H. (1989a). Decision analysis techniques for knowledge acquisition: Combining information and preference models using *Aquinas*. *International Journal of Man-Machine Studies*, in press.

BRADSHAW, J. M. & BOOSE, J.H. (1989b). Knowledge acquisition as CASE for knowledge-based systems. *Presentation at the Third International Workshop on Computer-Aided Software Engineering (CASE-89)*, London, England, July 17-21, 1989.

BRADSHAW, J.M. & BOOSE, J.H. (1989c). Recent work at Boeing on automating knowledge acquisition techniques. In R.R. Hoffman (ed.), *Knowledge Acquisition for Knowledge-Based Systems*. New York: Springer-Verlag, in press.

BRADSHAW, J.M. & BOOSE, J.H. (1989d). Knowledge representation for a Corporate Memory Facility. Boeing Computer Services Advanced Technology Center Technical Report BCS-CMF-TR2. NASA Contract No. NAS2-12108, 31 July.

BRADSHAW, J.M., BOOSE, J.H., COVINGTON, S.P., & RUSSO, P.J. (1989). How to do with grids what people say you can't: The application of decision analysis methods in *Axotl* and personal construct methods in *Aquinas* to design problems. *Knowledge Acquisition: An International Journal*, in press.

BRADSHAW, J.M., COVINGTON, S.P. & KIPERSZTOK, O. (1989). Process management and decision support tools for quality improvement. Boeing Computer Services Advanced Technology Center Technical Report, in preparation.

BRADSHAW, J.M., COVINGTON, S.P. & RUSSO, P.J. (1989). PIE: An application of *Axotl* to project selection decisions. Proceedings of the Boeing AI Conference, April.

BRADSHAW, J.M., COVINGTON, S., RUSSO, P., & BOOSE, J.H. (1989). KWACQ'ing with *DDUCKS:* Knowledge acquisition techniques for decision analysis using *Axotl* and *Aquinas*. *Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October.

BRADSHAW, J. M. & HOLTZMAN, S. (1987). Artificial intelligence and decision analysis. Presentation made to the Nineteenth Annual Meeting of the Society for Information Management, Seattle:,October 17.

BREESE, J.S. (1987). Knowledge representation and inference in intelligent decision systems. Doctoral dissertation, Department of Engineering-Economic Systems, Stanford University. Reprinted as Research Report 2, Palo Alto, California: Rockwell International Science Center.

ESHELMAN, L. (1988). MOLE: A knowledge acquisition tool for cover-and-differentiate systems. In S. Marcus (ed.), *Automating Knowledge Acquisition for Expert Systems*. Boston, Massachusetts: Kluwer Academic Publishers.

FEINSTEIN, D. (1989). Glass boxes: Lifting the veil on information. *Benchmark*, Spring, 12-15.

GAINES, B.R. (1988). Software engineering for knowledge-based systems. *Proceedings of the Second International Workshop on Computer-Aided Software Engineering (CASE-88).*

GRUBER, T.R. (1988). The acquisition of strategic knowledge. Unpublished Ph.D. dissertation. Department of Computer and Information Science: University of Massachusetts.

HOLTZMAN, S.H. & RUSSO, P.J. (1988). Activity graphs. Unpublished PIE project internal technical memo.

HOLTZMAN, S. H. (1989). *Intelligent Decision Systems*. Reading, Massachusetts: Addison-Wesley.

HOLTZMAN, S. & SEIVER, A. (1988). Decision analysis: A framework for critical care decision assistance. *International Journal of Clinical Monitoring and Computing*, in press.

HORVITZ, E.J. (1987). Reasoning about beliefs and actions under computational resource constraints. *Proceedings of the Thrid AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, July 10-12, 1987.

HORVITZ, E. J., BREESE, J. S., & HENRION, M. (1988). Decision theory in expert systems and artificial intelligence. *Journal of Approximate Reasoning*, in press.

HOWARD, R. A. (1988). Knowledge maps. *Management Science*, in press.

HOWARD, R. A. & MATHESON. J. E. (1981). Influence Diagrams. Reprinted in Howard & Matheson, 1984.

HOWARD, R. A. & MATHESON, J. E. (eds.) (1984). *Readings on the Principles and Applications of Decision Analysis*. Menlo Park, California: Strategic Decisions Group.

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

KEENEY, R. L. (1986). *Value-Driven Expert Systems For Decision Support*. Los Angeles: Systems Science Department, University of Southern California.

KELLY, G. A. (1955). *The Psychology of Personal Constructs*. 2 volumes. New York: Norton.

KLINKER, G. (1989). A framework for knowledge acquisition. *Proceedings of the Third Annual European Knowledge Acquisition Workshop*, Paris, France, July.

MARCUS, S. (1988). SALT: A knowledge acquisition language for propose-and-revise systems. In S. Marcus (ed.), *Automating Knowledge Acquisition for Expert Systems*. Boston, Massachusetts: Kluwer Academic Publishers.

MATHESON, D. (1988). Knowledge acquisition results on template development. Unpublished PIE project internal technical document.

MATHESON, J.E. (1988). Using influence diagrams to value information and control. *Proceedings of the Conference on Influence Diagrams for Decision Analysis, Inference, and Prediction*, Berkeley, California, University of California at Berkeley, May 9-11.

MCNAMEE, P. & CELONA, J. (1987). *Decision Analysis for the Professional—With Supertree*. Redwood City, California: Scientific Press.

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

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

OLMSTED, S. M. (1982). SUPERTREE - decision tree processing program. Menlo Park, California: Strategic Decisions Group.

RUSSELL, S. (1985). The compleat guide to MRS. Stanford Knowledge Systems Laboratory, Report No. KSL-85-12. Stanford, California: Stanford University Computer Science Department.

SHACHTER, R. D. (1986a). Evaluating influence diagrams. *Operations Research*, 34: 871-882.

SHACHTER, R. D. (1986b). DAVID: Influence diagram processing environment for the Macintosh. In *Proceedings of the RCA/AAAI Workshop on Uncertainty and Probability in Artificial Intelligence*.

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

SHEMA, D. B., BRADSHAW, J.M., BOOSE, J.H., & COVINGTON, S.P. (1989). Why a duck?: Design knowledge capture and alternative generation using possibility tables in *Canard. Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, October.

SIVARD, C., ZWEBEN, M., CANNON, D., LAKIN, F. & LEIFER, L. (1989). Conservation of design knowledge. *Proceedings of the 27th Aerospace Sciences Meeting of the American Institute of Aeronautics and Astronautics*, Reno, Nevada, January 9-12.

SMITH, J. E. (1988). On the representation and evaluation of influence diagrams. Unpublished PIE project internal technical document.

SULLIVAN, K. M. & SHULMAN, H.M. (1989). Cyclosporine and long-term follow-up care of the marrow transplant recipient: Clinical histories of chronic graft-versus-host disease, obliterative bronchiolitis and graft-versus-leukemia effect. *Transplant Proc.*, in press.

SZOLOVITZ, P. & PAUKER, S.G. (1978). Categorical and probabilistic reasoning in medical diagnosis. *Artificial Intelligence*, 11:115-144.

WELLMAN, M. (1986). Reasoning about assumptions underlying mathematical models. In J. S. Kowalik (ed.), *Coupling Symbolic and Numeric Computing*. Amsterdam: Elsevier.

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

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Structured Matching: A Task-Specific Technique for Making Decisions

Tom Bylander and Todd R. Johnson
Laboratory for Artificial Intelligence Research
Department of Computer and Information Science
The Ohio State University

Ashok Goel
School of Information and Computer Science
Georgia Institute of Technology

Abstract

All knowledge-based systems need to make decisions. Typically, these decisions involve selecting one choice out of a small number of alternatives based on a given set of parameters. The knowledge needed to make such decisions can often be expressed with a task-specific problem solving technique that we call structured matching, which integrates the knowledge and control for making a decision within a hierarchical structure. Structured matching unifies several independently-developed knowledge-based techniques, providing a common computational explanation of their success. We claim that structured matching is a ubiquitous and powerful technique for characterizing and constructing knowledge-based systems.

1 Introduction

Task-specific problem solving techniques are beginning to gain considerable attention from the artificial intelligence research community (Chandrasekaran, 1986; Clancey, 1985; McDermott, 1988). A task-specific technique *tractably* solves a limited range of problems by applying a specialized inference engine to a specialized knowledge base representation. If a problem can be solved by a task-specific technique, then the constraints of the technique can be useful for knowledge engineering, explanation, learning, etc.

In this paper, we describe a task-specific technique that we call *structured matching.* Structured matching is applicable to decision making tasks that involve selecting one choice out of a small number of alternatives based on a given set of parameters. In recognition, for instance, a set of confidence ratings can be the alternatives, and the absence or presence of features can be the parameters; in classification, the categories can be the alternatives, and the conditions differentiating between the categories can be the parameters.

Since the alternatives and parameters are typically known in advance, the knowledge engineering issue is how to organize and represent the knowledge for selecting the appropriate choices based on the values of the parameters. In structured matching, the knowledge and

*This work has benefited from many discussions with our colleagues at OSU's AI Lab, in particular B. Chandrasekaran, John Josephson, and Jack Smith. Of course, we are responsible for any remaining errors. This research has been supported by the the Air Force of Scientific Research, Grant AFOSR-87-0090, National Library of Medicine, NIH Grant 5 R01 LM 04298-03, the National Heart, Lung and Blood Institute, NIH Grant 1 R01 HL 38776-01, and the National Science Foundation, Grant CBT-8703745. Computer facilities were enhanced by gifts from the Xerox Corporation.

control needed for making such decisions are integrated within a hierarchical structure. Each node in the hierarchy corresponds to a different aspect of the decision and contains knowledge for directly mapping the results of its children nodes into a choice on the subdecision. The leaf nodes correspond to the parameters; the root node selects the final choice for the decision.

Structured matching has many desirable characteristics: it is computationally tractable, it is qualitative, it facilitates knowledge acquisition and explanation, and it explicitly represents decision-making knowledge. Thus, structured matching is clearly a task-specific technique.

Our understanding of structured matching has evolved from experimental research on knowledge-based systems. MDX, a medical diagnostic system in the domain of cholestatic liver disease, used structured matching to determine the likelihood of diseases (Chandrasekaran and Mittal, 1983). MDX's technique was incorporated into CSRL, a language for classificatory problem solving using classification hierarchies (Bylander and Mittal, 1986). Chandrasekaran (1986) called this technique hypothesis matching, and proposed it as an independently useful task-specific technique (a "generic task"). This led to the development of HYPER, a tool for building problem solving modules that measure the fit of a hypothesis to a situation (Johnson et al., 1989). Structured matching is a generalization of hypothesis matching.

Other knowledge-based research has independently developed techniques that implement structured matching. *Structured matching unifies these techniques, providing a common computational explanation of their success. Consequently, we claim that structured matching is a ubiquitous and powerful technique for characterizing and constructing knowledge-based systems.*

In this paper, we first describe structured matching informally, followed by a formal definition of the task and strategy of structured matching. We then describe how structured matching is implemented in the HYPER tool, showing how HYPER corresponds to the formal definition. Finally, we present examples of structured matching in several knowledge-based systems and compare it with other task-specific techniques.

2 Overview

All knowledge-based systems need to make decisions, e.g., perform classification or recognition. Often, these decisions have three characteristics:

> a small number of alternatives to select from,
>
> several parameters (including perhaps previous decisions) that are known to be relevant to the decision, where each parameter can take a value from a small number of values, and
>
> the need to make the same decision over and over again in different situations.

For instance, diagnostic systems often need to rate the confidence of a malfunction, e.g., whether the malfunction is Definite, Probable, Possible, or Ruled-Out. Diagnostic systems also need to decide what malfunctions should be considered. Typically, to make these decisions, several pieces of evidence, such as the presence or absence of symptoms

and previous decisions about malfunctions, need to be taken into account. If evaluating the presence of the malfunction is a routine part of a system's problem solving, then it makes sense to explicitly encode the knowledge for making the decision.

2.1 Simple Matching

One way to encode knowledge for making a decision is to directly associate patterns of parameters with the choices, for example:

$$param1 \wedge param2 \wedge (param3 \vee param4) \Rightarrow malfunction1 \text{ is Probable}$$

Since each such rule associates one pattern with one choice, additional knowledge (additional rules and an interpreter) is needed to encode the knowledge for a decision problem, i.e., to associate every possible pattern with a choice. We call this technique *simple matching* and the rules and interpreter that apply this technique to a particular decision problem, a *simple matcher*. Simple matching has several desirable features.

Explicit Representation of Decision-Making Knowledge. The patterns are directly associated with the choices.

Knowledge Acquisition. Simple matching has a simple knowledge acquisition strategy (Bylander and Smith, 1985). First, a decision to be made needs to be identified. Then, the parameters and choices of the decision should be listed. Finally, each pattern of values for the parameters should be associated with one of the choices. These steps can be performed by querying the domain expert or by analyzing protocols of the expert's problem solving. In addition, the simple matcher should be tested and refined using real cases (Sticklen et al., 1987).

Explanation. A decision made by a simple matcher is easy to explain. The parameters and their values that led to the choice can be stored in memory when the decision was made and retrieved whenever an explanation is required. If there are ordering dependencies between the patterns of a simple matcher, the parameters that caused the other patterns to fail can also be remembered for later retrieval. Note, though, that this only explains how the knowledge is used to make the decision, not how the knowledge itself is justified.

Qualitative Reasoning. Much of human problem solving appears to be qualitative in character. Simple matching can be used to encode the qualitative decision-making judgments of human experts.

2.2 Structured Matching

While simple matching has many attractive features, it also has a major drawback: simple matching is computationally intractable because the number of possible patterns grows exponentially with the number of parameters. Structured matching resolves this difficulty by allowing for a hierarchical structure in which the parameters of a simple matcher can be the results of other simple matchers.

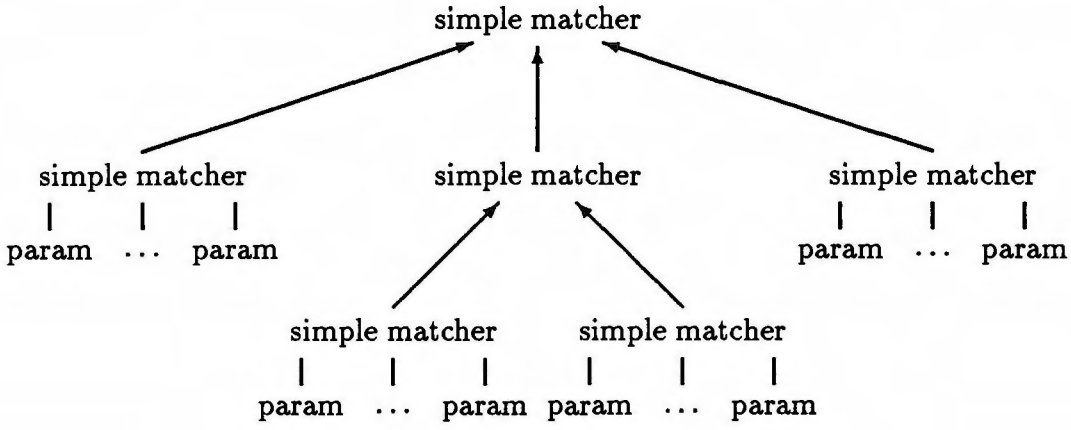


Figure 1: **Structured Matching**

Figure 1 illustrates structured matching. The parameters relevant to a particular decision are grouped according to domain criteria (e.g., causal connectedness, source of information), and each group is evaluated via simple matching. "Higher" simple matchers combine the results of "lower" simple matchers and, finally, the root simple matcher outputs a choice for the decision. The result of a lower simple matcher can be a recommendation for its parent or a "subdecision" on some feature of the parent's decision. The structure of the simple matchers can be a simple tree or a tangled hierarchy. However, there are limits on how much "tangledness" is allowed if computational tractability is to be maintained.

In the MDX system (Chandrasekaran and Mittal, 1983), for example, structured matching is used for evaluating the presence of diseases. The confidence in a disease is determined using a hierarchy of evidence combination tables. For instance, MDX evaluates the presence of biliary stone based on the strengths of xray, clinical, and historical evidence, as shown in Figure 2. The tables in Figure 3 represent portions of the knowledge for biliary-stone and clinical-evidence. Each row in a table corresponds to a simple rule. The first row of biliary-stone means "if clinical-evidence is +3 and historical-evidence is T then the confidence in biliary-stone is +3" (MDX uses a discrete scale from −3 to +3). Evaluation begins at the top row of the table and returns the confidence value of the first row that matches the evidence. Each table performs simple matching, and a hierarchy of such tables performs structured matching.

Structured matching retains most of the advantages of simple matching, with the additional benefit of avoiding combinatorial problems. In structured matching, the decision-making knowledge does not directly associate parameter values with a choice, but it does explicitly represent intermediate choices and the patterns that lead from one level to the next. The knowledge acquisition methodology involves an additional step: partitioning the parameters according to the subdecisions made by the domain expert, i.e., determining the parameters of each simple matcher in the structured matcher (Bylander and Smith, 1985). For explanation of problem solving, the explanations of the simple matchers can be combined

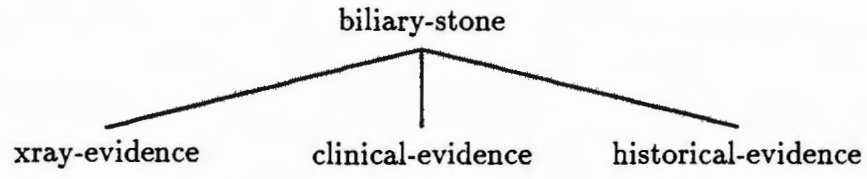


Figure 2: **Evidence Abstraction Hierarchy for biliary-stone**

xray-evidence	clinical-evidence	historical-evidence	biliary-stone
+3	+3	T	+3
...	...	...	...
0	+1	T	+2
...	...	...	...

cholangitis	colicky pain	nausea	clinical-evidence
T	T	T	+3
...	...	...	...
F	T	T	+2
...	...	...	...

Figure 3: **Truth Tables for biliary-stone and clinical-evidence**

to provide an explanation of the structured matcher (Johnson et al., 1989). In addition, the modularity of this technique results in good maintainability characteristics (Sticklen et al., 1987).

2.3 Relationship to General Formalisms

Above, we described structured matching in terms of rules, tables, and hierarchies. Thus, structured matching at first glance might appear to be a straightforward combination of three familiar ideas in artificial intelligence and computer science: production rules (Newell and Simon, 1972), decision tables (Cantrell et al., 1961), and hierarchical decomposition (Simon, 1981). However, since production rules and decision tables are general enough to be Turing-universal, they do not ensure computational tractability. Moreover, hierarchical decomposition, without additional constraints, does not guarantee tractability (Chapman, 1987). Structured matching addresses the problem of computational tractability by combining and constraining these general formalisms.

In structured matching, production rules are only permitted to match specific inputs from "below" to perform a specific action—making a choice. Decision tables are a general programming structure, essentially a *while* statement containing a generalized *if* statement

with arbitrary conditions and actions. Structured matching restricts the conditions of the table to be inputs from "below" and the actions to return a single output, with no iteration permitted. Structured matching applies a specific form of hierarchical decomposition—the parameters of a decision-making task and the knowledge to solve the problem are hierarchically partitioned.

Thus, structured matching incorporates a constrained version of these general formalisms. We believe that these constraints capture the essence of what makes a range of decision-making problems tractable to solve.

3 Definition

The previous section was an informal discussion of structured matching and its characteristics. In this section, we formally describe the functional specification of the *select-1-out-of-n task* and strategy of structured matching used to perform this task.

3.1 The Task

A problem solving task can be defined by specifying the information it takes as input and the information it gives as output. Let $C = \{c_1, \ldots, c_n\}$ be a set of n *choices*, where n is some "small" integer. Let $P = \{p_1, \ldots, p_m\}$ be a set of m *parameters*. Let $V = \{v_1, \ldots, v_q\}$ be a finite set of q *values*, where q is some "small" integer. Let d be a map from P to V, $: P \rightarrow V$, which assigns a value from V to each $p \in P$. Then $\{\langle p_1, d(p_1)\rangle, \ldots, \langle p_m, d(p_m)\rangle\}$ s the *input* for the select-1-out-of-n task.

For instance, P can be a set of assertions about a given situation, and V can be interpreted as a set of truth values, in which case $d(p)$ would be the truth value of the assertion p. Note that since q can be more than 2, this allows for specification of *incomplete knowledge*. For example, in the case that q is three, v_1 can be interpreted as true, v_2 as unknown, and v_3 as false. P can also be a set of variables and V can be interpreted as qualitative values, e.g., positive, zero, and negative. P can actually be a mixture of assertions, variables, etc., but for notational convenience, we use V as the range of values for all the parameters. This does not entail any loss of generality as long as V is large enough to accommodate the range of values for every parameter and as long as each parameter can have different interpretations of its values. For example, the values of p_1 might be interpreted as true, unknown, and false, while the values of p_2 might be interpreted as positive, zero, or negative.[1]

We define the select-1-out-of-n task of selecting one choice out of n alternatives as a four-tuple $\langle P, V, C, s\rangle$, where P, V, and C are the parameters, values, and choices (cf. (Goel et al., 1987)). s is a function from D to C, $s: D \rightarrow C$, where D is the set of all possible functions from P to V. For a given situation described by a specific function $d \in D$, s takes d as the input and gives a specific choice $c \in C$ as the output. The knowledge engineering problem is the construction of s so that appropriate choices are made computationally efficiently.

[1]Structured matching *per se* is intended to be an account of how decisions are computed, not how they are to be interpreted.

3.2 The Strategy

A structured matcher is a collection of simple matchers organized as a hierarchy. Different branches of the structured matcher evaluate different data, with each simple matcher mapping a limited number of inputs to a subdecision. The inputs of a simple matcher can be the results of other simple matchers or specific decision parameters or a combination of both.

Since simple matchers can be thought of as performing a smaller version of the select-1-out-of-n task, we describe them in that notation. A *simple matcher* is defined as a tuple $\langle P, V, C, table \rangle$, where P, V, and C are the parameters, values, and choices of the simple matcher. *table* is a data structure which associates each $d \in D$ with a specific choice $c \in C$. That is, the function $s: D \rightarrow C$ that a simple matcher performs is encoded in *table*. Evaluating a simple matcher is just finding the choice that is associated with the input data. The size of *table* is $O(q^m)$, where $m = |P|$ and $q = |V|$.[2] Since simple matchers are not intended to have an unbounded number of parameters, values, or choices, we use k to denote a "small" constant that limits the size of P, V, and C for any simple matcher. While it is hard to prescribe a specific value for k, our experience is that $k \geq 10$ can lead to problems in practice. Since k is a constant, the maximum size of *table*, and in general any simple matcher, is also a constant.

A *structured matcher* is recursively defined as a tuple $\langle P, V, C, \mathcal{P}, matchers, simple \rangle$, where again P, V, and C are the parameters, values, and choices, respectively. $\mathcal{P}$ is a tuple $\langle P_1, \ldots, P_r \rangle$, where $\{P_1, \ldots, P_r\}$ is a partition of P with rank r, $1 < r \leq k$, i.e., $P_1, \ldots, P_r$ are nonempty disjoint sets whose union is P. *matchers* contains the structured matchers corresponding to each P_j, $1 \leq j \leq r$. Of course, if $|P_j| = 1$, then no structured matcher is needed. *matchers* is a tuple $\langle x_1, \ldots, x_r \rangle$. If $|P_j| > 1$, then x_j is another structured matcher with P_j, V, and C as its parameters, values, and choices.[3] If $|P_j| = 1$, then $x_j = P_j$. *simple* is a simple matcher with r parameters, and with V and C as its values and choices.

This definition makes explicit one critical distinction between simple matchers and structured matchers. For simple matchers the number of parameters in P is bounded by k. For structured matchers, however, *only the rank* r of the partition of P is bounded by k.

Given an input $d \in D$, evaluating a structured matcher takes place in two steps. In the first step, the input for the simple matcher *simple* is determined by evaluating each x_j in *matchers*. If x_j is a structured matcher, then the jth input for *simple* is the value of x_j evaluated on input d. If x_j is not a structured matcher, but $x_j = P_j = \{p\}$, then the jth input value is $d(p)$. In the second step, the simple matcher *simple* is run with these r input values. The output of *simple* is the output of the structured matcher.

3.3 Guidelines for Use

Two guidelines follow from this definition of structured matching.

1. The functionality of a structured matcher is limited to making one decision based on data that is provided to the structured matcher. The problems of what decisions to

[2] q^m is only the upper bound on the size of table. In fact, it is often possible to encode a simple matcher much more efficiently, as we show in the discussion on the HYPER language.

[3] Again, using the same V and C for different structured matchers is a formal convenience. The interpretations of V and C can in general be different for different structured matchers.

make, and how to store and retrieve data are left to other problem solving processes or to knowledge engineers.

2. Structured matching is useful only if domain knowledge is available in the required form: each parameter and the output can only have a small number of possible values, the data must be grouped statically, and knowledge for mapping the parameters to subdecisions to the final decision must be available. If quantitative calculations or dynamic grouping of data are necessary, or if there is no source for knowledge acquisition, then structured matching should not be used.

These guidelines are important because they characterize the range of problems for which structured matching is appropriate.

3.4 Computational Complexity

A key issue in deciding on the utility of a problem solving technique is whether it is computationally tractable. Goel et al. (1987, 1989) have shown that structured matching is computationally tractable both for time and for "size," i.e., the size of the knowledge base needed to encode the structured matcher.

However, the formal definition requires that the parameters evaluated in different branches of the structured matcher be disjoint. What happens to the computational complexity of structured matching if this requirement is violated? Alternatively, what happens if the organization of simple matchers is a tangled hierarchy instead of a simple tree? If these restrictions are completely relaxed, then the worst case is exponential time complexity coupled with exponential size complexity. However, the worst case only happens if there are an exponential number of simple matchers in the structured matcher. Thus, a considerable amount of overlapping parameters among different branches or tangledness within the hierarchy can be tolerated without resulting in unacceptable performance.

3.5 Relation to Classification and Recognition

The formal definition of the select-1-out-of-n task and the strategy of structured matching describes the kinds of computations that structured matching performs, but does not provide a semantic account of the tasks for which structured matching is suitable. In this section, we give some feeling for how structured matching can be used for classification and recognition.

The tasks of classification and recognition are closely related to the select-1-out-of-n task. In one version of the classification task, we are given a set of categories and a set of data describing a specific entity (e.g., an object, a state, a situation, or a case), and the task is to determine the category to which the entity belongs. This task can be viewed as a select-1-out-of-n task, where the categories are the choices. In the recognition task, we are given a concept and a set of data describing a specific entity, and the task is to determine the degree of belief that the entity belongs to the concept. If the degree of belief is expressed qualitatively, (e.g., high, medium, low), then the recognition task can be viewed as a select-1-out-of-n task, where the belief values are the choices.

Since the strategy of structured matching does not impose any specific interpretation on the choices, it is potentially applicable to both of the above tasks. Whether or not

structured matching should be used depends on whether the decision-making knowledge meets the guidelines of structured matching described above.

Another version of the classification task is, given a set of categories and a set of data, determine the degree of belief in each category. The output of this task is not a single category, but the belief values of all the categories. This task can be decomposed into a set of recognition tasks. That is, determining the degree of belief in a category is an instance of the recognition task. Structured matching is potentially useful for this version of the classification task because each category can be associated with a structured matcher that determines the degree of belief in the category.

4 Implementation

HYPER (for HYPothesis matchER) is an example of a tool for building structured matchers. HYPER is a language for implementing a special version of structured matching that we call hypothesis matching, which applies to a version of the recognition task. The context is a hypothesis about a specific entity (e.g., a situation or a case) and a set of parameters whose values describe the entity. The hypothesis can take any one belief value in a n-valued belief system. Given a specific set of parameter values, i.e., given a specific description of the entity, the task is to determine a belief value for the hypothesis. The knowledge engineering issue is how to organize the domain knowledge so that the patterns of parameter values discriminate between the belief values.

The basic programming construct of HYPER is the *knowledge group* (KG). Each knowledge group represents a single simple matcher used for evidence combination in hypothesis matching. Figure 4 is a knowledge group named `BiliaryStone` for determining the likelihood of a stone causing cholestasis. `XrayEv`, `HistoricalEv`, and `PhysicalEv`, at the top of the first three columns, are the evidence used by the knowledge group to determine a likelihood. In this case, all three expressions refer to knowledge groups, but in general the expressions can be queries to any other problem solver capable of providing the appropriate information. The rows of the decision table are rules which associate a belief value with a particular pattern of evidence. For example, the second row can be read as "if `XrayEv` indicates `Likely` or higher, then the likelihood of `BiliaryStone` is the value of the `XrayEv` knowledge group". The `value` column can contain either a belief value (a member of a user defined sequence of symbols) or a call to another knowledge group. HYPER provides a small number of predicates for testing evidence. The `BiliaryStone` knowledge group contains only comparisons and "don't-care" (?) conditions. Knowledge groups are evaluated left to right, top to bottom. The result of the first rule (row) to match is used as the belief value of the knowledge group. If none of the rows match, then the `otherwise` expression is used to determine the belief value.

Figure 5 is the knowledge group for `PhysicalEv` from column 3 of `BiliaryStone`. In this knowledge group, the evidence is obtained by querying a database called `PATREC` (for PATient RECord) (Mittal and Chandrasekaran, 1984). The tests in column 2 of the table are true if the answer to the second query is equal to `colicky`. The first row can be read as "if cholangitis is present and the patient has colicky pain in the abdomen, then physical evidence for biliary stone indicates `HighlyLikely`."

BiliaryStone Knowledge Group
SubKGs: (XrayEv HistoricalEv PhysicalEv)

Expressions of table
1 - (Match XrayEv)
2 - (Match HistoricalEv)
3 - (Match PhysicalEv)

1	2	3	value
(ge Likely)	?	?	(Match XrayEv)
(ge Neutral)	?	(ge Likely)	Likely
(ge Neutral)	(gt Neutral)	(ge SlightlyLikely)	Likely
(ge Neutral)	?	?	Neutral
(lt Neutral)	(gt Neutral)	(ge Likely)	SlightlyLikely
(lt Neutral)	(le Neutral)	(ge Likely)	Neutral
(lt Neutral)	(gt Neutral)	(le SlightlyLikely)	SlightlyUnlikely
(lt Neutral)	(le Neutral)	(le SlightlyLikely)	Unlikely

otherwise Neutral

Figure 4: **Knowledge group for cholestasis caused by biliary stone**

PhysicalEv Knowledge Group
SubKGs: NIL

Expressions of table
1 - (Invoke ($ Patrec) Ask (QUOTE cholangitis) (QUOTE present?))
2 - (Invoke ($ Patrec) Ask (QUOTE abdomen) (QUOTE (pain quality)))
3 - (Invoke ($ Patrec) Ask (QUOTE vomiting) (QUOTE present?))

1	2	3	value
(eq yes)	(eq colicky)	?	HighlyLikely
?	(eq colicky)	(eq T)	Likely
?	(eq colicky)	?	SlightlyLikely
(eq yes)	?	?	SlightlyLikely

otherwise SlightlyUnlikely

Figure 5: **Knowledge Group for physical evidence of biliary stone**

This simple example shows how HYPER can be used to build a hypothesis matcher using domain-specific knowledge. This example also shows how qualitative decision-making judgments can be explicitly represented in HYPER. More complex hypothesis matching can be implemented by adding additional pattern combinations, knowledge groups, and hierarchy layers.

In some respects, HYPER improves on the formal definition of structured matching. The formal definition requires that every possible pattern in a simple matcher be represented. For the `BiliaryStone` knowledge group, this would entail $7^3 = 343$ patterns. However, the judicious use of `?` and tests like `ge` in columns 1–3, as well as the `Match` operation in the `value` column, reduces this to only 9 patterns. This is because only those patterns that can distinguish between belief values need to be encoded. Knowledge acquisition and explanation can also be done more efficiently.

HYPER also improves on the formal definition in the evaluation of a structured matcher. The formal definition requires that all the knowledge groups below `BiliaryStone` be evaluated. However, if the value of the `XrayEv` knowledge group is `Likely` or higher, the values of the other knowledge groups are not relevant. HYPER accommodates this interaction, so that a knowledge group is evaluated only if its value is needed in matching a pattern.

HYPER and its predecessor within CSRL have been used to construct recognition components of several knowledge-based systems in a variety of domains. For example, AUTOMECH performs malfunction diagnosis of automobile fuel systems (Tanner and Bylander, 1985); RED identifies red-cell antibodies for use in blood banks (Smith et al., 1985); WELDEX tests for defects in welds (Mahalingam and Sharma, 1985); NPP performs malfunction diagnosis and sensor validation of a nuclear power plant (Hashemi et al., 1986); CHEME performs malfunction diagnosis of a chemical processing plant that produces terephthalic acid (Shum et al., 1988); and JESSE models Japanese political decision-making in the domain of energy supply security (Sylvan et al., 1990).

We are continuing to work on several aspects of HYPER, in particular justifying and generating knowledge within KGs based on "deeper" knowledge and on chunking the results of problem solving. There is also work on a tool called RA (Recognition Agent) to succeed HYPER (Josephson and Smetters, 1989). RA will be one tool within an integrated set of task-specific tools for constructing knowledge-based systems.

5 Survey of Related Research

Despite the evident success of HYPER (and CSRL) in constructing practical structured matchers, we do not claim it is the single best language for structured matching. Many other knowledge-based systems covering widely varying domains perform structured matching and provide different constructs for its implementation. Below we present several such systems and illustrate how structured matching is used by each system. In each case, structured matching was developed independently without reference to the other systems. Structured matching unifies the techniques described below, providing a common computational explanation of their success. These systems also give strong evidence for the utility and ubiquity of structured matching for problem solving. In addition, we contrast structured matching with related techniques.

Major Criteria	Minor Criteria
1. Swollen hands	1. Myositis, mild
2. Sclerodactyly	2. Anemia
3. Raynaud's phenomenon or esophageal hypomotility	3. Pericarditis
4. Myositis, severe	4. Arthritis $\leq$ 6wks
5. CO diff capacity	5. Pleuritis
	6. Alopecia

	Definite	Probable	Possible
Requirements	4 majors Positive RNP antibody	2 majors & 2 minors Positive RNP antibody	3 majors
Exclusions	Positive SM antibody	No exclusions	No Exclusions

Figure 6: Criteria Table

5.1 Examples of Structured Matching

Samuel's work on checker playing programs provides an early example of structured matching (Samuel, 1967). To efficiently evaluate a board position, Samuel abandoned a monolithic evaluation function and introduced *signature tables*, which allow interrelated board features to be grouped and abstracted into higher level concepts. For example, 4 features can be mapped into a range of 5 values. These abstracted values then become inputs to another signature table and are further combined until a final decision is made. Each signature table maps from a set of features to a range of values using a simple table lookup. With a three level signature table scheme, Samuel was able to describe effects of complicated feature interactions and determine their contribution toward the overall board evaluation, which provided much of the leverage for his learning algorithm. The signature table technique closely corresponds to structured matching—each signature table corresponds to a simple matcher, while a hierarchy of tables is equivalent to a structured matcher.

Criteria tables (Kingsland and Lindberg, 1986) were used in AI/RHEUM (Lindberg et al., 1980), a knowledge-based system in rheumatology, to represent medical knowledge relating findings to a confidence in a disease. The criteria table in Figure 6 lists the major and minor findings relating to the diagnosis of mixed connective tissue disease. Following the findings are three diagnostic rules corresponding to three levels of confidence. Each rule describes the requirements needed for a diagnosis of the disease with the corresponding confidence. The findings list can contain references to other criteria tables. Therefore, a criteria table corresponds to a simple matcher and a hierarchy of tables corresponds to a structured matcher. The use of criteria tables in AI/RHEUM points to a natural generalization of structured matching in which individual simple matchers can be shared by different structured matchers, in this case because different diseases share some of the same high-level features. Also, the SEEK programs (Politakis and Weiss, 1984; Ginsberg et al., 1988) show how a form of learning called knowledge-base refinement can be performed on structured matchers.

```
Name: classic-esophageal-spasm
Isa: disease
Triggered-by: (confirmed chest-pain-when-eating)
Combining-function: IF (or (confirmed barium-swallow = spasm)
                           (confirmed manometrics))
                       THEN confirmed
                    IF (or (confirmed vasodilator-tx)
                           (confirmed nitroglycerin-tx))
                       THEN strongly-confirmed
                    IF (confirmed crescendo-pain-long-duration)
                       THEN supported
                    ...
```

Figure 7: **MUM Cluster for Classic Esophageal Spasm**

Gruber and Cohen (Gruber and Cohen, 1987) describe MUM, an architecture for managing uncertainty in medicine. MUM provides a declarative structure called a *cluster* for determining belief in a disease or "data interpretation." Clusters contain knowledge in the form of a combining function for mapping from evidence components to belief in the concept. For example, Figure 7 is a cluster for determining the presence of a classic esophageal spasm. The combining function uses findings and data interpretations, which can themselves be other clusters, to determine whether classic esophageal spasm is either disconfirmed, detracted, supported, strongly-supported, or confirmed. Thus, the combining functions within the clusters perform structured matching to determine confidence.

AQUINAS, a knowledge acquisition workbench (Boose and Bradshaw, 1987), represents simple matching knowledge using *rating grids.* A rating grid encodes knowledge which maps features to an element in a solution set. Figure 8 is a rating grid which maps programming language features to programming languages. The traits in this grid are rated from 1 to 5. Each cell in the grid represents the strength of a trait for the associated language. To use the grid, traits (Attitude, Availability, Application Area) of specified strengths are compared to the columns to find the solution (Ada, Lisp, Prolog, etc.) that best fits the specifications. AQUINAS points to a different way to encode and evaluate simple matchers, i.e., the conditions of the patterns are not evaluated as true or false, but in terms of closeness to the actual data.

In DSPL (Brown and Chandrasekaran, 1989), a tool for building routine design systems, a design plan is chosen from a set of competing design plans by using *sponsor-selectors.* Each plan has a *sponsor* responsible for determining the plan's suitability during the design process. The suitability, which is based on selected parameters of the design and the problem solving context, is measured on a symbolic scale such as (Perfect, Suitable, Don't-know, Not-suitable, Rule-out). In this way, sponsors are performing simple matching. The *selector* then uses the results of the sponsors and other data to select an appropriate design plan. Figure 9 displays the default selector in DSPL, which selects a "perfect" plan if there are any, else

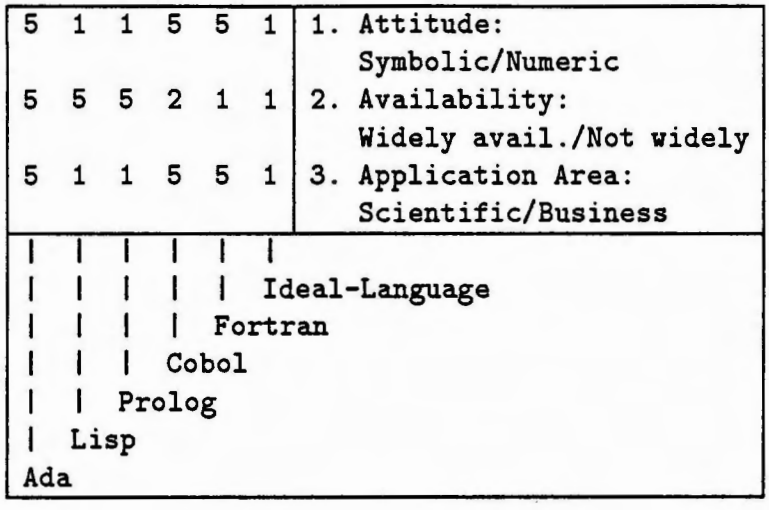


Figure 8: **Rating Grid in Aquinas**

```
(SELECTOR (NAME DefaultSelector)
          (BODY
               REPLY (if PERFECT-PLANS then
                         (DESIGNER-PREFERENCE PERFECT-PLANS)
                       else if SUITABLE-PLANS then
                         (DESIGNER-PREFERENCE SUITABLE-PLANS)
                       else NO-PLANS-APPLICABLE)))
```

Figure 9: **Default Selector in DSPL**

selects a suitable plan if there are any, else states that no plans are applicable. For more domain-specific selectors, it is possible to add domain knowledge that prefers one plan over others in case of ties. The selector is therefore a simple matcher for selecting one of a set of plans. Since the selector uses the results of other simple matchers (the sponsors), the sponsor-selector plays the role of a structured matcher.

Each of these examples provides different ways to encode simple matchers, but they all organize the simple matchers hierarchically as structured matchers to perform their decision-making tasks. The particular formalism that is used to encode simple matchers is not crucial as long as the advantages of efficiency, explicitness, knowledge acquisition, and explanation are achieved. What is important is that, in each case, simple matchers are limited to small decision-making tasks. Structured matching provides a means to perform larger decision-making tasks by composing simple matchers, yet still retaining the advantages of simple matching.

5.2 Techniques Similar to Structured Matching

Many artificial intelligence techniques share certain features of structured matching, but lack important defining characteristics. In this section we discuss some of these techniques and describe how they differ from structured matching.

The QBKG backgammon system employs a hierarchical evaluation function (Berliner and Ackley, 1982). Primitive observations are combined into ratings for concepts using various mathematical operators. These concepts are then combined into higher level concepts until everything is merged into a single heuristic value estimating the strength of a particular move. Throughout this process the values of all concepts are continuous—they are never mapped to discrete symbols. Berliner's evaluation function is similar in spirit to structured matching. A major part of both techniques is the abstraction of features through intermediate concepts. However, because Berliner's technique produces continuous values using continuous functions, patterns of data cannot be explicitly associated with specific values. This implies a more complex knowledge acquisition technique than that described for structured matching.

Many knowledge-based systems calculate the relative likelihood of a hypothesis by first representing, for each relevant datum or pattern, the "certainty" that the datum/pattern implies the hypothesis (and perhaps vice versa), and then combining the certainties for the data/patterns that appear in the actual case data. This technique is commonly used in diagnostic knowledge-based systems, e.g., the scores of PIP (Szolovits and Pauker, 1978), the evoking strengths and frequencies of INTERNIST-I (Miller et al., 1982), the weights of CASNET (Weiss et al., 1978), and the certainty factors of MYCIN (Shortliffe, 1976). Within each system, a global combining function performs the calculation, and numeric thresholds are used to provide interpretations of the results. This approach assumes that reasonable certainties can be acquired from domain experts (or a sample of cases), and that they can be combined independently. When these assumptions fail, structured matching can be used to explicitly represent dependencies between groups of data. Bayesian causal networks can also represent dependencies between data (Pearl, 1986), but like QBKG, require a more complex knowledge acquisition technique.

Decision trees differ from structured matching in that parameters are not partitioned over the branches in a decision tree. Instead the top node examines one parameter, with one branch for each possible parameter value. Each branch potentially considers all of the remaining $m-1$ parameters, each with q possible values. In the worse case, this can result in a decision tree of size $O(q^m)$. Also, decision trees do not represent groupings of parameters. However, decision trees are appropriate for implementing simple matchers because simple matchers only handle a small number of parameters (Goel and Bylander, 1989).

Heuristic classification (Clancey, 1985) characterizes the problem solving behavior of many knowledge-based systems as a three-part inference structure. (1) The data abstraction relation maps observations into qualitatively-valued parameters. (2) The heuristic match relation maps data to classificatory hypotheses. (3) The refinement relation indicates classificatory relationships between hypotheses. However, heuristic classification does not provide mechanisms that use the above relations. For example, structured matching is useful for one aspect of the heuristic match relation, that of mapping data to confidence in a classificatory hypothesis. However, EMYCIN (van Melle, 1979) with its certainty factors is an

alternative mechanism for implementing this behavior (as is done in HERACLES (Clancey, 1986)). The critical distinction is that heuristic classification describes the decisions that many knowledge-based systems make, while structured matching describes a mechanism that can make some of those decisions.

6 Summary

The search for task-specific problem solving techniques follows from the need to understand how intelligent problem solving is possible. Without sufficient constraints on how knowledge is organized and used, the problem solving can easily become an intractable process. Without explicit representation of knowledge, it is difficult to determine whether the problem solving is appropriate for a given task, and to acquire the needed domain knowledge. Explanation should also be tractable and relevant to the task.

To these ends, we have proposed a task-specific technique called structured matching. Structured matching can be summarized as follows:

Task Specification Given a small number of choices for a decision and a set of parameters relevant to making the decision, where each parameter can take a value from small number of values, determine a choice.

Form of Knowledge Patterns that map the parameters and results of other simple matchers to choices.

Organization of Knowledge A hierarchical organization of simple matchers, where each simple matcher makes a subdecision based on a small number of inputs. Each simple matcher specifies how selected parameters and the results of its children should be mapped to a choice. The root simple matcher makes the final decision.

Control As in HYPER, control can be initiated in a top-down fashion, i.e., each simple matcher calls on its children as needed in order to evaluate the conditions of a pattern. Information flows bottom-up.

Classification and recognition are examples of tasks for which structured matching is especially applicable. Whether structured matching should actually be used for these or other decision making tasks depends on whether the right kind of domain knowledge is available. If so, the use of structured matching offers many advantages. It can explicitly represent the qualitative decision-making judgments of experts. It facilitates knowledge acquisition and explanation. It is computationally tractable. Languages like HYPER can provide additional advantages over the formal definition by representing simple matchers more concisely and by evaluating structured matchers more efficiently.

The effectiveness of structured matching is empirically demonstrated by the number of systems that employ this technique as part of their problem solving. However, we would like to go further and provide a formal analysis of the kinds of information processing tasks for which structured matching is effective. We are currently working on adapting our model to address this issue.

Structured matching has emerged as a powerful and ubiquitous task-specific technique for knowledge-based systems. Its computational benefits are due to a coupling between an information-processing task and a technique. The technique of structured matching is specific to the select-1-out-of-n task, and it allows decision-making knowledge to be clearly represented and efficiently applied. We believe that this coupling between techniques and tasks will explain much about how intelligent decision making is possible.

References

Berliner, H. J. and Ackley, D. H. (1982). The QBKG system: Generating explanations from a non-discrete knowledge representation. In *Proc. National Conf. on Artificial Intelligence*, pages 213–216.

Boose, J. H. and Bradshaw, J. M. (1987). Expertise transfer and complex problems: Using AQUINAS as a knowledge acquisition workbench for expert systems. *Int. J. Man-Machine Studies*, 26(1):3–28.

Brown, D. C. and Chandrasekaran, B. (1989). *Design problem solving.* Morgan Kaufmann, San Mateo, CA.

Bylander, T. and Mittal, S. (1986). CSRL: A language for classificatory problem solving and uncertainty handling. *AI Magazine*, 7(3):66–77.

Bylander, T. and Smith, J. W. (1985). Mapping medical knowledge into conceptual structures. In *Proc. Expert Systems in Govt. Symposium*, pages 503–511, McLean, VA.

Cantrell, H. N., King, J., and King, F. E. H. (1961). Logic structure tables. *CACM*, 4(5):272–275.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30.

Chandrasekaran, B. and Mittal, S. (1983). Conceptual representation of medical knowledge for diagnosis by computer: MDX and related systems. In Yovits, M., editor, *Advances in Computers*, volume 22, pages 217–293. Academic Press, New York.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3).

Clancey, W. J. (1985). Heuristic classification. *Artificial Intelligence*, 27(3):289–350.

Clancey, W. J. (1986). From GUIDON to NEOMYCIN and HERACLES in twenty short lessons. *AI Magazine*, 7(3):40–60.

Ginsberg, A., Weiss, S. M., and Politakis, P. (1988). Automatic knowledge base refinement for classification systems. *Artificial Intelligence*, 35(2):197–226.

Goel, A. and Bylander, T. (1989). Computational feasibility of structured matching. *IEEE Trans. Pattern Analysis and Machine Intelligence.* To appear.

Goel, A., Soundararajan, N., and Chandrasekaran, B. (1987). Complexity in classificatory reasoning. In *Proc. Sixth National Conf. on Artificial Intelligence*, pages 421–425, Seattle.

Gruber, T. and Cohen, P. (1987). Design for acquisition: Principles of knowledge system design to facilitate knowledge acquisition. *Int. J. Man-Machine Studies*, 26(2):143–159.

Hashemi, S., Miller, D. W., Chandrasekaran, B., and Hajek, B. K. (1986). Expert system application to plant diagnosis and sensor validation. In *Proc. Power Plant Dynamics, Control and Testing Symposium*, Knoxville, TN.

Johnson, T. R., Smith, J. W., and Bylander, T. (1989). HYPER—Hypothesis matching using compiled knowledge. In *Proc. AAMSI Congress*, pages 126–130, San Francisco.

Josephson, J. R. and Smetters, D. (1989). Generic task toolset, Fafner release—beta test, including RA. Technical report, Lab. for AI Research, CIS Dept., Ohio State Univ., Columbus, OH.

Kingsland, L. C. and Lindberg, D. A. B. (1986). The criteria form of knowledge representation in medical artificial intelligence. In *Proc. Fifth World Conference on Medical Informatics*, pages 12–16, Washington, DC.

Lindberg, D. A. B., Sharp, G. C., Kingsland, L. C., Weiss, S. M., Hayes, S. P., Ueno, H., and Hazelwood, S. E. (1980). Computer-based rheumatology consultant. In *Proc. Third World Conference on Medical Informatics*, pages 1311–1315, Tokyo.

Mahalingam, S. and Sharma, D. D. (1985). WELDEX—An expert system for nondestructive testing of welds. In *Proc. Second Conf. on AI Applications*, pages 572–576, Miami Beach, FL.

McDermott, J. (1988). Preliminary steps toward a taxonomy of problem solving methods. In Marcus, S., editor, *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic, Norwell, MA.

Miller, R. A., Pople, H. E., and Myers, J. D. (1982). INTERNIST-I, an experimental computer-based diagnostic consultant for general internal medicine. *New England J. of Medicine*, 307(8):468–476.

Mittal, S. and Chandrasekaran, B. (1984). PATREC: A knowledge-directed database for a diagnostic expert system. *Computer*, 17(4):51–58.

Newell, A. and Simon, H. A. (1972). *Human Problem Solving.* Prentice-Hall, Englewood Cliffs, NJ.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288.

Politakis, P. and Weiss, S. M. (1984). Using empirical analysis to refine expert system knowledge bases. *Artificial Intelligence*, 22(1):23–48.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers II—Recent progress. *IBM Journal*, 11(11):601–617.

Shortliffe, E. H. (1976). *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York.

Shum, S., Davis, J., Punch, W. F., and Chandrasekaran, B. (1988). An expert system approach to malfunction diagnosis in chemical plants. *Computers in Chemical Engineering*, 12(1):27–36.

Simon, H. A. (1981). The architecture of complexity. In *Sciences of the Artificial*, chapter 7, pages 192–229. MIT Press, Cambridge, MA, 2 edition.

Smith, J. W., Svirbely, J. R., Evans, C. A., Straum, P., Josephson, J. R., and Tanner, M. C. (1985). RED: A red-cell antibody identification module. *J. Medical Systems*, 9(3):121–138.

Sticklen, J., Smith, J. W., Chandrasekaran, B., and Josephson, J. R. (1987). Modularity of domain knowledge. *Int. J. Expert Systems*, 1(1):1–15.

Sylvan, D. A., Goel, A., and Chandrasekaran, B. (1990). Analyzing political decision making from an information processing perspective. *American J. Political Science.* To appear.

Szolovits, P. and Pauker, S. G. (1978). Categorical and probabilistic reasoning in medical diagnosis. *Artificial Intelligence*, 11(1-2):115–144.

Tanner, M. C. and Bylander, T. (1985). Application of the CSRL language to the design of expert diagnostic systems: The AUTO-MECH experience. In Richardson, J., editor, *Artificial Intelligence in Maintainence*, pages 149–169. Noyes, Park Ridge, NJ.

van Melle, W. (1979). A domain independent production-rule system for consultation programs. In *Proc. Sixth Int. Joint Conf. on Artificial Intelligence*, pages 923–925, Tokyo.

Weiss, S. M., Kulikowski, C. A., and Amarel, S. (1978). A model-based method for computer-aided medical decision-making. *Artificial Intelligence*, 11(1-2):145–172.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A FRAMEWORK FOR THE COMPARATIVE EVALUATION OF KNOWLEDGE ACQUISITION TOOLS AND TECHNIQUES

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

ABSTRACT
The last Banff workshop identified the evaluation of knowledge acquisition (KA) tools, techniques and products as being a key concern. This paper presents and demonstrates the use of a framework for generating testable propositions to guide such empirical evaluation research and for integrating the findings of past, ongoing and future studies. By considering the tools and techniques used in KA as independent variables, it isolates two major categories of dependent variables and discusses empirical measures for them. Additionally, it examines four sets of moderating variables that bear upon the success of any KA activity: 1) human factors, 2) problem space characteristics, 3) system development approach, and 4) the organizational environment. The major obstacles impeding the formal comparison of KA techniques in controlled settings are also discussed.

1. THE NEED FOR RESEARCH COMPARING KNOWLEDGE ACQUISITION TECHNIQUES

Over the last three decades, the application of computer systems to organizational problems has gone through several stages of increased sophistication from transaction processing to managerial support systems. These stages have paralleled the advances in hardware and software technologies and the understanding of how to apply them to existing and new problems. Knowledge-based systems (KBS) are at the center of attention today. As is the case with any new information technology, the initial focus in KBS has been in demonstrating feasibility of system development and application, mostly in academia and in industrial research laboratories. Now that KBS feasibility has been demonstrated, it is important that researchers embark upon a carefully planned research program to better understand how to efficiently and effectively develop such systems if commercial viability is to be assured.

A key aspect of KBS development is knowledge acquisition (KA). It is the general name given to the process of eliciting, acquiring, and representing knowledge consisting of descriptions, relationships and procedures in a specialized domain of interest. A large number of KA techniques and tools now exist for the acquisition and representation of expertise (Welbank, 1987; Olson and Reuter, 1987; Boose, 1988), and new ones are being proposed. Potential users of KA techniques and tools need assistance in evaluating these alternatives, and in combining and matching them to their particular needs. This concern has prompted calls for research (e.g., Fellers, 1987) on the following: 1) Is there one best elicitation technique for KA? 2) If not, what is the best combination of techniques? 3) Which techniques are most suitable under which circumstances? and 4) What skills are required in order to utilize each of these techniques?

To date a substantial amount of effort in the KA area has been spent on tool building and subsequently applying them to particular problems to show their feasibility. However, most of the empirical evidence about the economic value and capabilities of alternative KA methods has been collected by their developers or proponents. It is only recently that independent evaluations of the usability of some automated KA tools have taken place (e.g., Kitto, 1988). It is important to continue such investigations and conduct them in an organized rather than *ad hoc* manner. In calling for the empirical investigation of KA tools and techniques, we propose a framework to guide such research efforts and to integrate the findings of current studies in the field. In addition, the framework offers definitions for the variables identified and suggests ways to operationalize them. The next two sections describe the components of the framework. This is followed by a review of the few studies that have attempted to empirically investigate alternative KA techniques. Lastly, the major measurement, operationalization, and confounding problems plaguing such KA research are discussed.

2. PROPOSED RESEARCH FRAMEWORK

Prior to outlining the components of the framework shown in Figure 1, it is important to explain its purpose. A framework assists in organizing and structuring the entities of interest. Frameworks serve to: 1) highlight the major dimensions of the area under study, as well as indicating those which are not important, in order to guide researchers, 2) provide a scheme for comparing and integrating the findings of independent studies, and 3) suggest a common language for investigators to talk about the critical issues of the field (Treacy, 1985).

The objective of the KA process is to develop *efficiently* an operational KBS of *high quality* and functionality. Efficiency is measured by the time, cost and effort expended by the expert(s) and knowledge engineer(s), as well as other organizational and technological resources, utilized during the KA phase. The quality of what is produced by the KA process is a difficult concept to operationalize. It has to be a valid representation of the expertise source(s) modelled and it has to be of value within a particular application domain. These variables will be discussed in more detail in the next section.

In Figure 1, quality of the KBS and efficiency of the KA process, are labelled as "dependent variables" because they are influenced by a number of factors that are important in any KA effort. The independent variable posited to be most influential on the behavior of the dependent variables is the KA technique or tool used. However, the effects of the KA technique used, as well as the decision made to select a particular KA technique, will be moderated by the influence of a host of other variables. A moderator variable is "any variable which when systematically varied 'causes' the relationship between two other variables to change" (Stone, 1978, p. 26) and in this case is the relationship between the dependent variables and the KA technique. The moderator variables included in the framework are the attributes of 1) the expert, 2) the knowledge engineer, 3) the problem domain, 4) task type, 5) the system development approach and 6) the organizational environment in which the KBS is being developed. These are presented diagrammatically in Figure 1, while their components are detailed in Table 1, 2, and 3 (all figures and tables are placed at the end of the paper.)

Note that in Figure 1 the quality of the KBS and the efficiency with which it is developed are directly influenced both by the choice of KA method used and values of the moderator variables shown on the left side of the figure. For example, the quality of KA will be influenced both the by the knowledge level of an expert and by the ability of the KA method utilized to capture knowledge in the particular application domain. Similarly, the amount of previous KA experience of a knowledge engineer will influence the quality of the KBS developed through the selection of a KA method that is most familiar to the knowledge engineer and his ability to best utilize this method. It is also expected that the moderator variables could directly influence the dependent variables in certain cases. For example, the knowledge level of the expert modelled will influence the quality of the KBS and the past experience of the knowledge engineer will affect the efficiency of the KA process.

The next section describes the framework in more detail, and suggests ways in which the variables represented in the framework could be operationalized and measured.

3. COMPONENTS OF THE FRAMEWORK

3.1 Dependent Variables

The dependent variables are summarized in Table 1.

3.1.1. Quality Of The KBS

The three major questions relating to the evaluation of quality are 1) when is measurement to take place? 2) what is to be measured? and 3) what is the standard of comparison?

Hayes-Roth et al. (1983) suggest that an iterative series of evaluations as the KBS is being developed would provide a good measure of the dependent variables. Measurements of quality can be taken at any one of several phases in the KBS development process. The knowledge acquired can be evaluated 1) at its initial point of capture, 2) after it has been converted into an initial prototype knowledge base, 3) during or after further iterations to refine an operational KBS, or 4) as a fully functional KBS. However, in situations where quality measures requiring comparisons of a KBS to some standard (e.g., percentage of correct diagnoses and percentage of correspondence to expert behavior) are desired then data collection should be done at the final, fully functional stage.

In measuring the quality of a knowledge base or KBS, four different aspects of quality could be considered (Benbasat and Dhaliwal, 1989):

1) the *representational validity* of the KBS;
2) the *value* or *perceived benefits* obtained from its use;
3) the *usability* and *acceptance* of the KBS; and
4) the *design architecture* of the KBS.

3.1.1.1 Validity - Benbasat and Dhaliwal (1989) define validity of a KBS as the degree of its representational homomorphism with the source system it is supposed to model (e.g., a human expert). Thus, comparisons to the source expert, after whom a KBS is modelled, could be used to determine the validity of a KBS. Validity has two components: behavioral and structural. Behavioral

validity involves comparing the output or recommendations of the KBS to that of the expert. Structural validity is concerned with the similarities in the knowledge structures and reasoning process used by the KBS and the expert. Johnson (1983) presents a somewhat similar view with quality of KBS defined as having outcome and reasoning components. He further suggests that KBS which reason using the same process as the expert and arrive at the same conclusion can be considered to be superior to those that arrive at the same conclusion using a different line of reasoning.

3.1.1.2 Value - This could be measured as: 1) benefits, actual or perceived, that users obtain when the KBS is applied to their problems, and 2) quality as determined by a group of experts in the context of a problem to which the KBS is applied, or measured objectively based on quantitative criteria of performance. Examples of specific measures would be the degree of user satisfaction with the KBS especially with regards to how well it performs, and the actual tangible cost savings that the KBS generates when put into use.

3.1.1.3 Usability and Acceptance - Hayes-Roth et al. (1983) argue that the ultimate test of the quality of a KBS is the level of acceptance of the system by the users, and the extent to which they depend on it for consultation. Other measures include evaluations of different aspects of the interface such as ease-of-use, naturalness, and flexibility of using the KBS as compared to interaction with the human expert.

3.1.1.4 Design Architecture - It is expected that the KBS will go through changes and modifications on a regular basis. Therefore, the quality of KBS design architecture is an important aspect of quality. Quality standards include measures of completeness, modularity, cohesiveness of subunits, coupling between subunits, consistency between rules, ease of maintenance, and ease of expansion. Such measures can be obtained by means of objective benchmark testing or through the judgement of a panel of experts.

3.1.2 Efficiency Of The KA Process

The efficiency of KA is associated with the resources expended in the development of a knowledge base. The effort, cost, and time of the expert(s) and the knowledge engineer(s) are the major determinants of efficiency. Since both knowledge engineers and source experts are scarce and costly resources, and impose constraints on the speed with which KBS are developed, minimizing their individual efforts and their time of interaction is clearly desirable. This has been the major motivation for building automated KA tools which reduce the need for interaction and communication between the two parties. However, since a central objective of KBS development is to build high quality KBS, it makes little sense to develop KA tools and techniques that optimize the efficiency of the development process at the expense of quality. This may be especially relevant to the use of automated KA tools that capture efficiently only a partial subset of the total set of different types of knowledge required for a particular KBS application.

3.2 Independent Variable: Knowledge Acquisition Techniques

These are summarized in Table 2. The discussion which follows is based on the view of the KBS development process as being composed of three distinct task phases: capturing, structuring and representation. Only the first two together constitute KA, while representation involves transcribing the expertise acquired into an appropriate machine manipulable form. The

capturing and structuring phases can be thought of as iterative cycles which are closely intertwined. Capturing is associated with the techniques used to elicit knowledge from the expert, while structuring involves formalizing the elicited knowledge to make sense of what is captured.

3.2.1. Knowledge Capturing

KA techniques can be classified in a multitude of ways. These include: 1) automated vs. manual (nonautomated) methods, i.e., the degree to which the technique is implemented as a computer program (Boose, 1988), 2) direct methods which require experts to report knowledge that they can directly articulate, and indirect methods which do not rely on the articulation abilities of experts but instead focus on inferring what the expert "must have known" from his responses (examples of indirect methods include multi-dimensional scaling, hierarchical clustering, repertory grid analysis and the use of inductive learning algorithms (Olson and Reuter, 1987)), and 3) techniques focused on specific task or problem solving methods or those that are more generalized. Boose (1988) classifies techniques according to application tasks (e.g., diagnosis, classification, debugging, repair, design, scheduling, etc.), and problem solving methods (e.g., heuristic classification and heuristic construction).

The above categories are not orthogonal. Most KA techniques possess both direct and indirect components. The procedure of collecting verbal protocols is an example of the direct aspect as the expert is required to describe his expertise in action. However, the procedure of analyzing protocols using KRITON (Linster, 1988) which transforms the protocols into operator-argument structures can be termed as indirect since the KA tool is attempting to deduce from the protocols the knowledge structure possessed by the expert. Similarly, those aspects of KSSO that elicit the entities and bi-polar attributes of an expert's conceptual structure can be viewed as being the direct component, and the analysis of the ratings in the repertory grids using FOCUS, PRINCOM, and ENTAIL as the indirect component.

Automated KA tools have had the most impact on the 'indirect' or analysis component of KA by reducing the tediousness of performing such analysis and by greatly augmenting the types of analysis possible. Nonautomated techniques focus more on the direct aspects of eliciting responses from the expert and can take two forms. They are either *descriptive* or *observational* in nature. For example, prior to the application of indirect KA techniques like similarity-based learning (e.g., INSTIL: Kodratoff and Manago, 1987) and explanation-based learning (e.g., ACES: Pazzani, 1987) the set of examples needed has to be directly acquired either by an expert's description or through observation of an expert's task performance. Relating this to the validity of KBS developed, it may be hypothesized that direct methods of KA will more closely model human expertise than indirect models.

Descriptive techniques facilitate the capture of expertise as described by the expert. This is commonly done by the use of interviews and by creating formalized computer based description languages for the expert's direct use, e.g., KA techniques that are applications of personal construct theory elicit responses in terms of the entities, attributes and ratings constituting the expert's conceptual structure.

As the expertise being sought is by nature cognitive rather than physical, techniques of cognitive observation are required in addition to visual observation techniques. Such techniques include the collection of expert performance protocols on real or hypothesized cases and the accumulation of experts' recall, sorting, rating or scaling responses.

KA tools also differ in their range of application, domain specificity, and exploitation of problem solving methods. Some, like OPAL (Musen et al., 1987) in the cancer treatment domain, bank heavily on the characteristics of one particular domain to guide the KA activity. Others, like ETS (Boose, 1985), are contrived to be domain independent. Similarly, while systems like SALT (Marcus, 1987) exploit the peculiarities of a particular problem solving method, others like KRITON are more generally applicable to an assortment of problem solving methods.

3.2.2 Knowledge Structuring

KA tools and techniques can be classified according to the structuring formalism used. This relates to classification according to the type of knowledge elicited, such as facts, conceptual structures, procedures, weights of evidence, justifications, etc. While some structuring formalisms, e.g., production rules, frames and semantic networks, have a knowledge representation tradition, others have been adopted from other fields such as decision analysis (decision trees), information requirements analysis and data modelling (entity relationship diagrams), and psychotherapy (repertory grids).

Besides facilitating the organization and management of the knowledge captured, structuring formalisms also serve a useful purpose in assisting knowledge engineers form a quick and correct conceptualization of the KBS domain.

Although attempts have been made to directly compare KA capturing techniques (e.g., Burton et al., 1987), there has been no attempt to directly compare KA structuring formalisms or KA techniques that are based on different formalisms. Additionally, work relating to the integration of these capturing and structuring techniques into circumstance-specific repertoires of KA techniques has also been scarce.

3.3 Moderator Variables

The moderator variables are summarized in Table 3.

3.3.1. The Human Components

3.3.1.1 Attributes of the Source Expert - Sources of expertise can be classified into two general categories: authentic sources and reconstructed sources (Johnson, 1983). Human experts exemplify the former category. Reconstructed knowledge (or expertise) flows from textbooks, non-expert introspection, databases, normative mathematical models, etc. Feigenbaum (1978) advances a somewhat identical taxonomy comprising expertise private to a human expert and that which is in the public domain. The KA problem is mostly restricted to knowledge transfer from such authentic or private sources (i.e., human experts), despite the fact that many existing KBS utilize reconstructed expertise.

Some attributes of experts that impact KA are 1) the stage of expertise development attained, 2) cognitive individual differences of the experts (such

as cognitive styles), and 3) personality variables. Of these, the first two can be considered to have the most impact on the selection and use of KA techniques and are discussed here.

The stage of development of an expert has a direct impact on his or her ability to articulate expertise and thereby influences both the choice of the KA technique selected and the dependent variables of Figure 1. The stage of expertise development variable can be related to the three stages of expertise development of Anderson's (1982) theory of skill acquisition.

This theory views the skill learning (or expertise) process as a sequence of phases termed the declarative phase, the knowledge-compilation phase and the procedural phase. During the declarative phase, the novice learns from instruction or from the observation of the consequences of actions undertaken. His initial performance involves the operation of general strategies utilizing declarative knowledge to guide performance. The knowledge-compilation phase involves the conversion of the slow and conscious declarative knowledge interpretations into faster compiled procedures. At this stage, the individual is beginning to develop an initial mental model. As the associations between the separate components become stronger, there is no longer a need to consciously initiate each separate action. In the final procedural phase, the newly acquired models are refined and tuned through the process of over-practicing and reinforcement to the point of automation and unconsciousness, often termed the tacit dimension of expertise (Polanyi, 1966).

It is the expertise of experts at phase three that is usually sought to build KBS. However, such experts are not conscious of or able to describe their thought processes. This implies that stage two experts may provide better descriptions of their expertise than stage three experts, although the expertise will be of a lower quality. A more important implication is that since stage three experts are not able to describe their expertise directly, descriptive techniques of KA such as interviewing experts may have limited potential as compared to observational techniques.

Although much is known of the influence of individual differences (such as cognitive styles) on human cognitive functioning, the wisdom from various studies in the decision support system domain, is that they are difficult constructs to operationalize and often suffer from the lack of an adequate and coherent theoretical basis (Huber, 1983). However, some individual differences are especially relevant to KA. An example is the ability of an individual to perceptually and logically disembed a whole into its constituent parts. This is can be viewed as a measure of an expert's ability to analyze and explain the components of his knowledge to others. Acquiring knowledge from experts lacking this ability would be a more onerous task and would effect the KA method selected.

3.3.1.2 Attributes of the Knowledge Engineer - The knowledge engineer's responsibilities include administering the KA technique, analyzing and structuring the expertise extracted, and implementing the expertise as a KBS. The expertise level of the knowledge engineer is a significant factor and has a few aspects including the knowledge engineer's familiarity with the domain, experience with the KA technique(s) being used, and past general experience in

KA. All these can be hypothesized to have a potential impact on the process of KA.

Another variable involved is the role of the knowledge engineer in the application of the various KA techniques and tools. For example, in using the techniques of participant observation, teachback interview, and structured interviewing, the knowledge engineer has a more active role than in the case of techniques such as unstructured interviewing, verbal protocol and repertory grid analyses. These differing roles call for different knowledge engineering skills and impact the KA process differently. For example, Woodward (1987) posits that the role of knowledge engineering is undergoing redefinition with the harder, quantitative side of knowledge engineering giving way to the qualitative side of designing and modelling. Fellers (1987) suggests that skills and abilities, such as effective communication, patience, diplomacy, and conceptual skills, are required for successful KA. Thus, the use of a particular KA technique, such as unstructured interviewing, will dictate the need for a different set of knowledge engineering skills from those required for using an automated tool such as the KSSO.

3.3.2 Problem Space Characteristics

3.3.2.1 Attributes of the Application Domain - Several criteria could possibly be used to distinguish among application domains. These include levels of complexity, degree of risk or uncertainty, number of alternatives and attributes, size of the search space, and levels of structure. Of these the existence of structure in a domain is singled out as being of importance in KA. According to Johnson (1984): "In well-structured problems, reconstructed methods of reasoning and authentic methods of reasoning may be quite similar since the number of ways for a task to be done is small. In more ill-structured domains, however, the two types of methods are often quite different, and authentic methods of reasoning become an important source of expert level algorithms for completing the task....."

Based on Keen and Scott-Morton's (1978) assertions that structure often reflects knowledge, and structure exists only to the extent that it is perceived, Warbelow (1986) posits that both the extent of underlying "deep structure" (Chomsky, 1971) in a domain and the extent to which it is specified, and thereby perceived, will impact KA. The underlying deep structure of a domain specifies the relationships in its problem space. Thus, while chess has high deep structure, weather forecasting possesses low deep structure. Perceived structure on the other hand, is defined by the extent to which experts in the domain have a well-defined and specified body of knowledge with which to work. KA would be most difficult when both the specificity of the knowledge and the actual existence of deep structure are low, as in the domain of natural language interpretation. However, in a domain such as accounting, both deep structure and specificity of the domain knowledge is high, thus knowledge acquisition would be relatively simpler.

3.3.2.2 Nature of the Task/Problem Solving Method - Within a specified domain, the nature of the task selected for KBS development will affect the choice of KA technique. Hayes-Roth et al. (1983, p. 14) classify knowledge engineering tasks into the following classes: interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, and control. Boose (1988) maps these onto the two problem solving methods of *heuristic classification* and *heuristic configuration*, and superimposes the

various KA tools onto this classification. This is based on the observation that most KBS tasks are strongly associated with particular types of problem solving methods, and that KA tools should concentrate on specific tasks or problem solving methods (Musen et al. 1987).

For example, Chandrasekaran (1983) proposes that a domain (e.g., medical problem solving) gets naturally decomposed into knowledge substructures each of which specializes in one type of problem solving (e.g., diagnosis). Each of these substructures in turn further gets decomposed into small knowledge sources of the same problem-solving type, but specialising in different concepts in the domain (e.g. clinician, pathologist, radiologist). Thus, identification of the knowledge substructures for the different task types can define a body of knowledge for a domain. This body of knowledge even if developed in rudimentary form can assist both in the development of automated KA tools, and in guiding the knowledge engineer in what KA approach to take.

3.3.3 System Development Approach

The two commonly used approaches to KBS development are rapid prototyping and the traditional life cycle approach (Weitzel and Kershberg, 1989). The former is most commonly used, although some KA techniques (such as unstructured interviewing, protocol analysis, and sorting) and tools (such as KSSO and Acquinas) are more suited to the latter approach which does not require a prototype system to be constructed until after the bulk of the knowledge has been acquired.

While it is common wisdom in KBS development that KA should be undertaken independent from implementation details, more common is the situation where early commitment to an automated KA tool necessitates the use of the prototyping approach. This points to an interesting sub-area for KA techniques research: how the use of KA techniques and tools impact the various steps of the KBS development process.

3.3.4 Organizational Environment

The resources that an organization puts into the development of a KBS (in terms of the time and cost of source experts and knowledge engineers) and consequently the commitment that the organization has in the project will depend on several factors. These include the competitive and strategic importance of the KBS to the organization, past organizational experience with KBS use and development, the technological culture in the organization, and the level of management support for the KBS development. The more prevalent these factors are in an organization, the more likely that the project will be successful. In an organization that is experimenting with KBS technology by undertaking a trial development, it can be expected that the KA method used will be one that produces a working prototype utilizing the smallest amount of resources. On the other hand, in organizations where the importance and commitment to KBS development is greater, the KA technique selected will be one that is more specifically appropriate for the particular application and multiple techniques may be employed. This may be at the expense of a longer KA period to achieve a KBS quality level commensurate with the importance of the project. Similarly, organizations with more KBS development experience and strong technological cultures can be expected to be using more complex KA techniques.

4. A REVIEW OF KA TECHNIQUE EVALUATION RESEARCH

Empirical research in evaluating KA techniques is at an early stage. The paucity of such work may reflect the prevailing high interest in developing and demonstrating the feasibility of KBS. However, as the field matures we would expect increased attention to be paid to research work that will more efficiently and effectively generate knowledge for the development of KBS.

Table 4 summarizes the various studies that evaluated one or more KA techniques in terms of the variables presented in the framework in Figure 1. A brief review of these studies is presented here.

Michalski and Chilausky (1980) compared two KA techniques ("learning by being told" and "learning by examples") in the context of developing a KBS for soybean disease diagnosis. They found that inductively learned rules (using the AQ11 system based on the Aq algorithm) outperformed rules acquired through interviewing an expert plant pathologist by generating correct diagnosis 97.6% versus 71.8% of the time. However, the study lacked internal validity. There was little control for the nature and order of questioning during the interviews and the test cases were not unbiased or randomly selected.

Grover (1983) compared four different methods of KA. These were 1) forward scenario simulation (interview/walkthroughs), 2) goal decomposition (traditional problem reduction), 3) procedural simulation (protocol analysis), and 4) pure reclassification (frame analysis). His results indicated that walkthroughs and frame analysis were of greatest value.

Littman (1986) utilized video protocol methodology to study the KA activities of six expert knowledge engineers. While offering no recommendation as to which KA technique is best, he concluded with the following suggestion: "the process of empirically studying the techniques that knowledge engineers use to extract mental models of domain experts is a potentially useful enterprise."

Henrion and Cooley (1987) compared two approaches (rule based and decision analysis based) of performing KA for developing diagnostic KBS for the treatment of root disorders in apple trees. The rule based approach used inference nets as the knowledge structuring formalism; the decision analytic approach utilized influence diagrams (Bayes' belief nets). They found that while the rule based approach required less effort initially, it required more extensive testing since the expert was more likely, as compared to the Bayesian approach, to encounter combinations of events that were not considered explicitly during design. This was because in construction of the decision analytic model, the use of conditional distributions to uncover diagnostic rules forced the expert to consider the impact of all possible combinations of evidence for each problem source. They also noted that the two approaches utilized different attitudes towards human reasoning under uncertainty. The rule based approach considered the expert's performance as an ideal to be emulated and discrepancies were viewed as a sign of a deficiency. However, the decision analytic approach considered it as being possibly unreliable, in the light of psychological evidence of the limitations of human reasoning under uncertainty, and sought to improve upon it using normative theory of the application area. This suggests that the latter approach may be better suited to KBS developments (for reasoning under uncertainty applications) based on reconstructed sources of knowledge or in

situations where a truly experienced expert is non-existent or difficult to obtain.

Messier and Hansen (1987) utilized the prototyping approach together with four KA techniques -- expert interviewing, protocol analysis, distilling knowledge from reconstructed sources (EDP audit materials of the Big Eight accounting firms) and expert walkthroughs of KB in the development of a KBS for evaluating the adequacy of data processing audit controls. They note that while protocol analysis provided some insight into the decision making behavior of computer auditing specialists, it failed to uncover appropriate production rules for their rule-network. This could reflect two things. Firstly, protocol analysis as a KA technique could be suited to uncovering only certain types of knowledge (e.g., the order of reasoning, or the alternatives or attributes considered) but not deductive rule-type knowledge. Secondly, it could reflect a representational mismatch between the KA method used (protocol analysis) or the knowledge structuring formalism used (rule network) and the domain or task knowledge.

Burton et al. (1987) in an experimental study directly compared four KA techniques: interviews, protocol analysis, laddered grid (goal decomposition), and multi-dimensional scaling based on card sorting. Knowledge was acquired from 32 undergraduate geology students, with each subject being exposed to two KA techniques each in a separate one-pass KA session. The domain selected was the geological identification of igneous rock and the dependent variables used were time taken for the KA session, time taken to code the transcripts into rules, number of rules elicited, number of clauses constituting the elicited rules, and completeness of the rule set. Two sets of psychometric tests were administered to the subjects to determine characteristics of their personality (introvert-extrovert) and cognitive style (field dependent-field independent). Their results indicate that protocol analysis takes longer to perform and analyze than the other techniques but yet retrieves substantially less knowledge. Introverts took longer time in interviews than extroverts but produced more rules and clauses for a fixed amount of knowledge acquired. Field dependent subjects had difficulty with the laddered grid technique as they found it hard to disembed spatial information in the goal decomposition diagram that had to be drawn. Field dependent subjects produced more rules and clauses than their field independent counterparts when using protocol analysis. The laddered grid and card sort-multidimensional scaling techniques were as efficient as the more common interview and protocol analysis techniques.

Shaw and Woodward (1987) in validating the KITTEN and KSSO tools, conducted a set of experiments in the domains of spatial interpolation techniques for producing contour maps, and trouble shooting and maintenance of valves for oil and gas pipelines. This provides a useful preliminary evaluation of the automated acquisition of personal constructs as a KA technique. The experiments tested the extent to which domain experts agreed with each other and with themselves at a later time, based on knowledge acquired from them using the KSSO tool. The experts also indicated their agreement with the outputs of structuring tools that were applied to the knowledge acquired from them. Their findings show that the KA tool performed satisfactorily in terms of the similarity of constructs elicited from different experts as well as the reliability of constructs generated by the same expert at different points of times.

Kitto (1988) contrasted the Acquinas and KNACK tools in terms of their ability to be used directly by a domain expert with minimal assistance from a knowledge engineer. The dependent variables used -- evaluations of usability and interface features -- correspond to the usability and acceptance aspect of KBS quality that was discussed earlier. Her findings conclude that the knowledge engineer is still essential to the KA process for the use of these tools. This suggests that while reducing or eliminating the role of the knowledge engineer has been one of the objectives for building some KA tools, it is possible that the influences of the moderating variables, such as expert's familiarity with the tool, may have led to a new role for knowledge engineers. Kitto also found that in the use of these tools experts were often unsure as to which terms in their domains constitute concepts, which others are characteristics, and which are specific instantiations of these characteristics. Our experience with the use of KSSO for identifying the entities and attributes for an audit control application supports this observation. These could possibly be the result of mismatches between the KA tool and the type of domain or task (a moderating variable).

In a nonempirical study, Linster (1988) compared KRITON to other automated tools. These included ETS and Acquinas which are geared towards the creation of functional knowledge-bases, KITTEN and KSSO which are structural analysis tools, SALT, OPAL and PROTEGE which are specialized knowledge editors, and KPT which is a modelling support tool. Based on this, he derived a set of design specifications for second generation KA tools and suggested that the techniques seem to have evolved enough to be integrated into a larger tool that is not limited to any particular task type, domain, problem solving method, or phase of the KA process.

This review of KA techniques research (as summarized in Table 4) serves to highlight several factors. Firstly, there have been few direct comparisons of the techniques. Secondly, the few studies that have been done have lacked adequate experimental rigor and control. Thirdly, most of the studies have not accounted for the effects of the moderator variables. Fourthly, the disparate nature of the studies reflects the need for a common research framework to integrate these findings. Additionally, it is encouraging to note from this review that both KA techniques and the moderator variables influence the dependent variables. This is as predicted by the framework and indicates the benefits of having such a scheme to guide research efforts.

The framework could also be advantageously used to generate a set of testable propositions. While the framework can motivate the generation of such propositions, evidence for them has to come from theory or prior empirical studies. The following is an *example of a subset of propositions* that can be generated for comparing two KA techniques: manual protocol analysis and the use of an automated tool (KSSO). These have been selected solely for illustrative purposes:

Relating to KBS Quality (Validity):
P1: Protocol analysis will elicit knowledge that is less valid, in relation to the source expert, than that elicited by KSSO.
Support: According to theories of skill acquisition and the "paradox of expertise", highly skilled experts have difficulty in self-articulating their

expertise directly. KSSO overcomes this by having the expert incrementally focus on and describe particular aspects of his conceptual structure.

Relating to KA Efficiency:
P2: KA with KSSO will be more efficient than protocol analysis.
Support: Automated techniques attempt to be more efficient by minimizing or eliminating the role of the knowledge engineer and by computerizing knowledge structuring and analysis.

Relating to Expert Attributes:
P3: KSSO will require experts with more analysis skills, while protocol analysis will require experts with greater verbalization skills.
Support: KSSO requires the ability to decompose an application domain in terms of its constituent entities and attributes; protocol analysis relies on think-aloud verbalization.

Relating to Knowledge Engineer Attributes:
P4: Knowledge engineering skills required for using KSSO are different than those required for protocol analysis.
Support: Protocol analysis requires coding, analysis and interpretation skills in examining the protocols, while with KSSO knowledge engineers need communication skills in guiding domain experts in the its use.

Relating to Domain/Task/Problem-Solving Method:
P5: KSSO is more suited for domains where the expertise is largely in declarative form, while protocol analysis is more suited for domains where procedural expertise is more dominant.
Support: Messier and Hansen (1987) report that protocol analysis is more appropriate for eliciting procedural knowledge.

Relating to System Development Approach:
P6: Protocol analysis is not as conducive to the prototyping approach as compared to KSSO.
Support: In protocol analysis extensive and time consuming interpretation of the protocols is required to formulate a complete expertise specification before a prototype can be implemented.

Relating to Organizational Environment:
P7: Organizations with less extensive KBS experience and exposure will prefer using KSSO to protocol analysis.
Support: Automated tools reduce the need for organizational expertise in building KBS.

5. DIFFICULTIES OF KA TECHNIQUES RESEARCH

The lack of formal comparisons of KA techniques in controlled settings was mentioned previously. This is not surprising since there are a number of difficulties in conducting such studies. They are discussed in this section.

One major drawback in investigating the impact of the KA techniques and tools on the dependent variables is controlling for the mediating influences of the other variables. An example is controlling for the attributes of the knowledge engineer in a study comparing two KA techniques. A sample of 20 experts are chosen and 10 of them are randomly assigned to one technique, the rest to the other. The following provides an indication of some aspects of

this problem: if only one knowledge engineer is used for acquiring knowledge from all experts across treatments (different KA tools and techniques), the knowledge engineer will learn as he proceeds. Thus each subsequent KA process is expected to improve in quality, thereby confounding the results. Alternatively, if 20 different knowledge engineers are assigned to the 20 different experts, then part of the quality of the final KBS developed could be attributed to the differences between the knowledge engineers. There is no simple solution for such problems. Schemes for controlling them can be suggested, for example, rotating knowledge engineers among subjects by assigning a different knowledge engineer to each phase of the KA process, i.e., a Latin square design. While, each scheme has its advantages in terms of controlling for learning effects and averaging out individual knowledge engineer differences, on the whole none of them provides a clearly better solution than the original alternatives.

Another drawback is the issue of operationalization of the dependent variables, especially the quality of KBS variable. Firstly, there is the question of at which stage should the measures be taken. Secondly, there is the question of what measures are reasonable, most importantly in situations where multiple KBS have been developed using different KA techniques. The use of any measure enumerating the number of rules, such as in the Burton et al. (1987) study, is flawed. A greater number of rules can imply both that more knowledge has been captured as well as it has been captured in a poorly structured form. There is also no standard measure of what constitutes a single rule as rules can have both multiple premises (IF) and action (THEN) clauses. Moreover, as is demonstrated by the use of variables in Lisp and Prolog, rules themselves can have multiple instantiations. In addition to rules, the Burton et al. (1987) study also used the number of clauses constituting the elicited rules as one of a set of dependent variables. This also suffers from the same weaknesses as using the number of rules.

The third aspect is that of how the quality measures should be obtained. Self-evaluation by source experts is hindered by the paradox of expertise. Evaluation by other experts and a panel of judges can provide such measures, but experts differ and there are issues of selection bias involved in using panels. For example, a panel of alternative experts could initially be asked to provide a measure of the overall quality of a KBS, next they would be asked for separate evaluations of the reasoning and recommendations. However, only the overall quality and the recommendation evaluations can be used directly in the analysis. This is because since experts differ significantly in their decision logic, reasoning evaluations by other experts could be meaningless.

Another difficulty is in distinguishing between what the expert contributes and the quality of the KA tool itself. For example, the poor quality of knowledge acquired could be attributed to both the expertise level of the expert and/or utilizing a KA tool of poor quality. Other problems inherent in undertaking empirical KA research include measuring and controlling for the variability of experts (Shaw and Woodward, 1987), suppressing the significant novelty effects that are inevitable considering that exposure to KBS and KA tools is still limited, the statistical inference difficulties associated with small samples sizes that are inevitable considering the cost and time necessary to develop such systems and the lack of experts willing to volunteer in formal studies, and generalizability problems due to the variety of KBS application tasks and domains.

6. SUMMARY

Knowledge-based systems have become part of the information technology portfolio of many large organizations. Knowledge acquisition to develop such systems is still considered to be a bottleneck and a major cost element. While a multitude of KA tools and techniques have been proposed, research efforts in evaluating these techniques are still in their infancy. This paper has proposed a framework, which includes a set of variables to be investigated, to guide investigators in the KA field. It is hoped that the framework would help in integrating the findings of ongoing studies, identifying areas in need of investigation, and in general, providing a common language to drive research in the empirical investigation of KA methods.

BIBLIOGRAPHY

Anderson, J.R. (1982). Acquisition of Cognitive Skill. *Psychological Review*, Vol. 89, pp. 369-406.

Benbasat, I. and Dhaliwal, J.S. (1988). A Framework for the Validation of Knowledge Acquisition. *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, AAAI, Banff, (November), pp. 1.1-1.18.

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

Boose, J.H. (1988). A Survey of Knowledge Acquisition Techniques and Tools. *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, AAAI, Banff, (November), pp. 3.1-3.23.

Burton, A.M., Shadbolt, N.R., Hedgecock, A.P., and Rugg, G. (1987). A Formal Evaluation of Knowledge Elicitation Techniques for Expert Systems: Domain 1. *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Reading University, (September), pp. D3.1-D3.20.

Chandrasekaran, B. (1983). Towards a Taxonomy of Problem Solving Types. *The AI Magazine*, (Winter/Spring), pp. 9-17.

Chomsky, N. (1971). *Aspects of the Theory of Syntax*, Cambridge, MA: MIT Press.

Fellers, J. W. (1987). Key Factors in Knowledge Acquisition. *Computer Personnel*, Vol. 11(1), (May).

Feigenbaum, E. A. (1978). The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering. *Proceedings from the AFIPS Conference of the 1978 National Computer Conference*, 47, pp. 227-240.

Gaines, B.R. and Shaw, M.L.G. (1981). New Directions in the Analysis and Interactive Elicitation of Personal Construct Systems. In *Recent Advances in Personal Construct Technology*, Shaw, M.L.G., (ed.),. London: Academic Press, pp. 147-182.

Grover, M.D. (1983). A Pragmatic Knowledge Acquisition Methodology. *Proceedings from the International Joint Conference on AI*, Karlsruhe, West Germany, pp. 436-438.

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

Henrion, M. and Cooley, D.R. (1987). An Experimental Comparison of Knowledge Engineering for Expert Systems and for Decision Analysis. *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, pp.471-476.

Huber, G.P. (1983). Cognitive Styles as a Basis for MIS and DSS Designs: Much Ado About Nothing. *Management Science*, Vol. 29(5).

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. (1984). The Expert Mind: A New Challenge for the Information Scientist. In Bemelmans, M.A. (ed.), *Beyond Productivity: Information System Development for Organizational Effectiveness*, Amsterdam: Elsevier Sciences Publishing.

Keen, P. and Scott-Morton, M. (1978). *Decision Support Systems: An Organizational Perspective*, Reading, MA.

Kitto, C.M. (1988). Progress in Automated Knowledge Acquisition Tools: How Close are we to Replacing the Knowledge Engineer. *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop, AAAI*, Banff, (November), pp. 14.1-14.13.

Kodratoff, Y. and Manago, M. (1987). Generalization in a Noisy Environment: The Need to Integrate Symbolic and Numeric Techniques in Learning. *International Journal of Man-Machine Studies*, Vol. 27(2), pp. 181-204.

Linster, M. (1988). A Critical Look at KRITON: Moving from First to Second Generation. *Proceedings of the Third Knowledge Acquisition for Knowledge-Based Systems Workshop, AAAI*, Banff, (November), pp. 17.1-17.19.

Littman, D.C. (1986). Modelling Human Expertise in Knowledge Engineering: Some Preliminary Observations. *Proceedings of the First Knowledge Acquisition for Knowledge-Based Systems Workshop, AAAI*, Banff, (November).

Marcus, S. (1987). Taking Backtracking With a Grain of SALT. *International Journal of Man-Machine Studies*, Vol. 26(4), pp. 383-398.

Messier Jr., W.F. and Hansen, J.V. (1987). A Case Study and Field Evaluation of EDP-XPERT. Working Paper, (July).

Michalski, R.S. and Chilausky, R.L. (1980). Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soyabean Disease Diagnosis. *Policy Analysis and Information Systems*, Vol. 4(2), (June).

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), pp. 105-121.

Olson, J.R. and Reuter, H.H. (1987). Extracting Expertise from Experts: Methods for Knowledge Acquisition. Technical Report No.13, Cognitive Science and Machine Intelligence Laboratory, Univ. of Michigan, (June).

Pazzani, M.J. (1987). Explanation-Based Learning for Knowledge-Based Systems. *International Journal for Man-Machine Studies*, Vol. 26(4), pp.413-434.

Polanyi, M. (1966). *The Tacit Dimension*, Garden City, NY: Doubleday.

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

Stone, E. (1978). *Research Methods in Organizational Behavior*, Glenview, Illinois: Scott, Foresman and Company.

Treacy, M.E. (1985). Toward a Cumulative Tradition of Research on Information Technology as a Strategic Business Factor. Working Paper, Center for Information Systems Research, Sloan School of Management, MIT, (April).

Warbelow, A. (1986). Knowledge Acquisition for Expert System Development. Unpublished Research Proposal, Harvard University.

Weitzel, J.R. and Kershberg, L. (1989). Developing Knowledge-based Systems: Reorganizing the System Development Life Cycle. *Communications of the ACM*, Vol. 32(4), (April), pp. 482-488.

Welbank, M. (1987). Knowledge Acquisition: A Survey and British Telecom Experience. *Proceedings of the First European Workshop On Knowledge Acquisition for Knowledge-Based Systems*, Reading University, (September).

Woodward, J.B. (1987). Hitting A Moving Target (The Training of Knowledge Engineers). *Proceedings of the Second Knowledge Acquisition for Knowledge-Based Systems Workshop*, AAAI, Banff, (October).

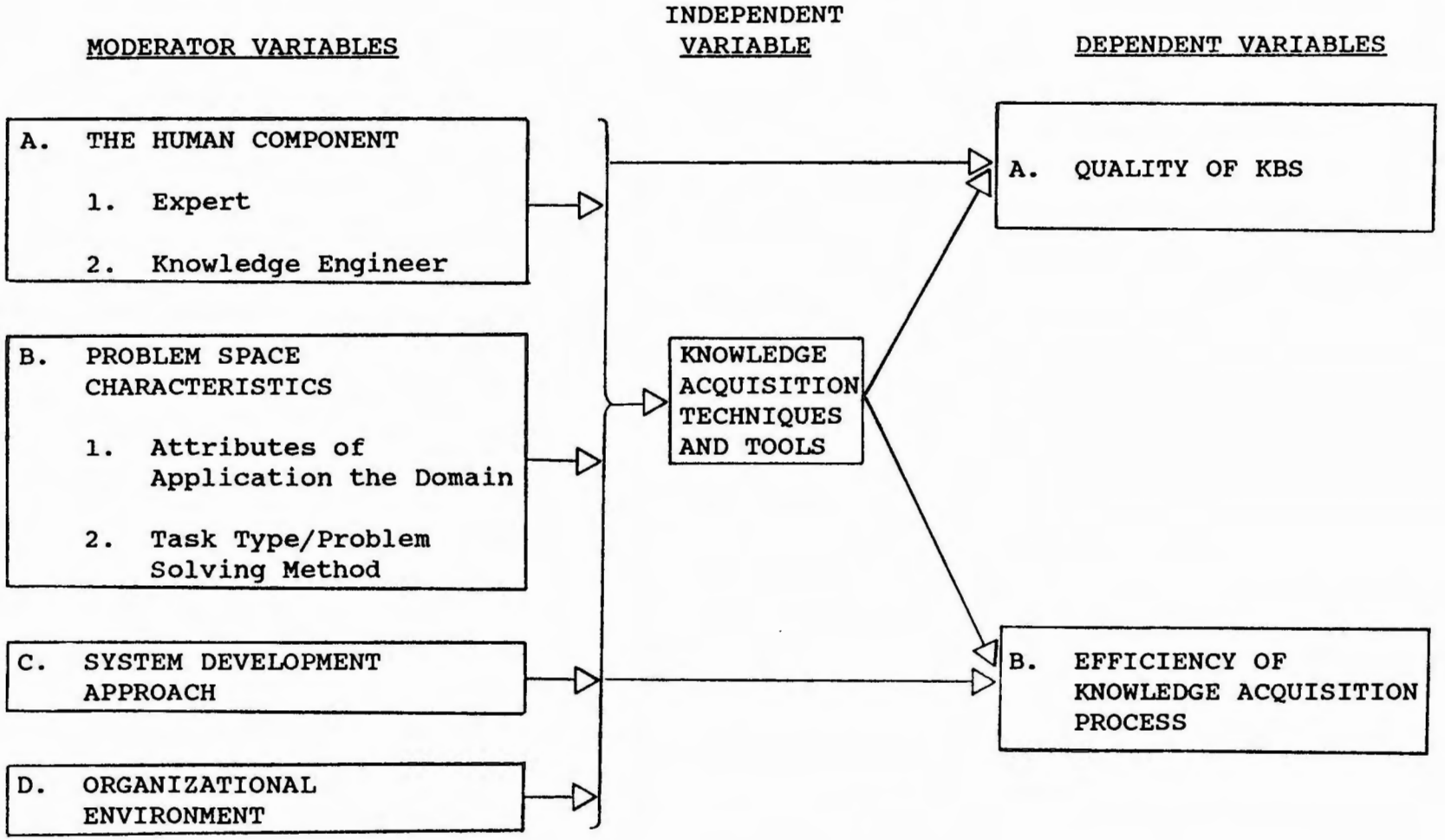


FIGURE 1
A RESEARCH FRAMEWORK FOR INVESTIGATING KNOWLEDGE ACQUISITION TECHNIQUES

QUALITY OF KBS
1) Representational Validity 2) Economic value or perceived benefits of KBS 3) Usability and Acceptance of KBS 4) Design architecture quality

EFFICIENCY OF KA PROCESS
1) Time, cost, and effort spent by expert(s) 2) Time, cost and, effort spent by knowledge engineers

TABLE 1: DEPENDENT VARIABLES

KNOWLEDGE ACQUISITION TECHNIQUES AND TOOLS
1) Methods for Capturing Expertise i. Automated vs. Non-Automated ii. Direct vs. Indirect iii. Task/Problem Solving Specific vs. General 2) Knowledge Structuring Formalisms i. Rules ii. Repertory Grids iii. Schemas/Frames iv. Networks (Semantic Nets) v. Decision Trees/Tables

TABLE 2: INDEPENDENT VARIABLE: KNOWLEDGE ACQUISITION TECHNIQUES AND TOOLS

THE HUMAN COMPONENTS

EXPERT

Source of expertise:

1) Authentic: Human expert(s)

   Characteristics of Human expert(s)

   a) Stage of expertise development
   b) Cognitive styles
   c) Personality variables

2) Reconstructed: Integration of Multiple Sources

KNOWLEDGE ENGINEER

1) Domain knowledge
2) Technical knowledge
3) Past KA experience in general
4) Past KA experience in specific KA methods

PROBLEM SPACE

ATTRIBUTES OF THE APPLICATION DOMAIN

1) Degree of inherent structure
2) Degree of perceived structure
3) Degree of uncertainty

TASK TYPES

1) Interpretation
2) Prediction
3) Diagnosis
4) Design
5) Planning
6) Monitoring
7) Debugging
8) Repair
9) Instruction
10) Control

PROBLEM SOLVING METHODS

1) Heuristic Classification
2) Heuristic Configuration

SYSTEM DEVELOPMENT METHODOLOGY

1) Prototyping
2) Traditional Life Cycle Methods

ORGANIZATIONAL ENVIRONMENT

1) Management Support
2) Past Experience in KBS development
3) Technological Culture
4) Strategic Purpose (e.g. competitive value of KBS to organization)

TABLE 3: MODERATOR VARIABLES

Studies	KA Techniques	Moderator Variables	Dependent Variables	Results/Comments
Michalski & Chilausky (1980)	- Interviewing - Inductive Learning	Not Considered	- Percentage of correct diagnosis generated	- Inductive learning (learning from examples) performed better than interviewing (learning by being told): 97.6% vs. 71.8% correct diagnosis as compared to the performance of a human expert
Grover (1983)	- Walkthroughs - Goal Decomposition - Protocol Analysis - Frame Analysis	Not Considered	- Knowledge engineer or system developer's opinion of the value of KA techniques	- Frame analysis and walkthroughs were of greatest value.
Littman (1986)	- KA techniques not directly compared. Focus was on the KA behavior of knowledge engineers	- Experienced knowledge engineers - Goal structure of knowledge engineers	- Assessment of actual knowledge and heuristics used by knowledge engineers	- Attempted to identify knowledge and heuristic used by knowledge engineers to develop process models as basis for simulating KA activities
Henrion & Cooley (1987)	- Inference Nets (rule based) - Influence Diagrams (Bayesian belief nets)	Not Considered	- Expert's time to develop KBS	- Inference nets (rule based) required less effort initially, but required more extensive testing

TABLE 4: SUMMARY OF STUDIES EVALUATING KA TECHNIQUES

Studies	KA Techniques	Moderator Variables	Dependent Variables	Results/Comments
Messier & Hansen (1987)	- Interviews - Protocol Analysis - Expert Walk-throughs	- Human and reconstructed expertise sources	- Knowledge engineer or system developer's opinion of the quality of knowledge acquired	- Protocol analysis had limited usefulness for certain types of knowledge
Burton, et al., (1987)	- Interviews - Protocol Analysis - Goal Decom-position - Card-sorting and Multi-Dimensional Scaling	- Introvert and extrovert personality - Field inde-pendent/de-pendent cognitive styles	- Time taken for capturing knowledge - Time taken for coding knowledge into rules - Number of rule clauses - Completeness of rule set (using an expert panel)	- Protocol analysis takes longer and yields less knowledge than other techniques - Introverts need longer interviews but generate more knowledge than extroverts - Field dependent subjects have dif-ficulty with goal decomposition - With protocol analysis field dependents generate more rules than field independents - Goal decomposition and card sorting can be as efficient as interviewing and protocol analysis

TABLE 4 (CONTINUED)

Studies	KA Techniques	Moderator Variables	Dependent Variables	Results/Comments
Shaw & Woodward (1987)	- KITTEN/KSSO	- Type of experts: academic researchers (pseudo-experts) vs. true experts	- Consensus between experts - Consensus and reliability over time - Meaningfulness of rules entailed	- Construct reliability was 100% for true experts and 50-83% for academic researchers (pseudo-experts) - Rating reliability was similar for both groups - Meaningfulness of rules from the automated KA tool was high
Kitto (1988)	- ACQUINAS - KNACK	- Role of knowledge engineer in the use of the tools	- Usability by domain experts - Quality of interface of the KA tools	- Large number of options in an integrated workbench like ACQUINAS can prove confusing to the novice user (domain expert or inexperienced knowlege engineers) - Experts are unsure of which terms in the domain are "concepts", which are "characteristics", and which are specific examples of characteristics - Knowledge engineer's role has not been diminished by automated KA tools
Linster (1988)	- KRITON - ETS/AQUINAS - KSSO/KITTEN - ORAL/PROTEGE - SALT - KPT - ROGET	Not Considered	Non-Empirical study	- KRITON features are compared with each of the other tools to derive specifications for second generation KA systems

TABLE 4 (CONTINUED)

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Hypothesis testing: an appropriate task for knowledge-based systems†

B.Z. Feldman, P.J. Compton, G.A. Smythe
Garvan Institute of Medical Research
St Vincents Hospital
Sydney
NSW 2010

ABSTRACT

We emphasise the distinction between insight and knowledge. Insight is the activity in which we recognise that some particular knowledge makes sense of some aspect of reality, while knowledge is something we make up to express our insights. From this distinction it follows that knowledge independent of insight may lead to error and that an individual's knowledge even in a single domain is liable to be inconsistent. We propose therefore that knowledge-based systems are used more properly for ensuring that the growth of knowledge is consistent, than as oracles. Moreover there is a major need for such systems. We describe a prototype for a medical research domain and detail initial results obtained with the system.

A PHILOSOPHY OF KNOWLEDGE AND KNOWLEDGE ACQUISITION

The conventional explanation of problems in knowledge engineering, particularly in the maintenance phase of expert systems, is that experts fail to report adequately on their underlying knowledge when asked to explain how they reach some conclusion (Waterman, 1986). We have suggested that experts do something quite different when they are asked why they reach given conclusions (Compton, Horn, Quinlan and Lazarus 1988; Compton and Jansen 1989). What they do is justify their conclusions. They identify the data and reasons which will persuade the person that they are communicating with, that in fact they are right. The material they produce to justify their position will vary with the context in which the dialogue takes place. For example the expert will present different arguments when dealing with a peer, a trainee in the area, or a lay person. It will also depend on the expert's recent reading, problems, emotional state etc. etc.

This method of communicating one's knowledge is not a regrettable habit that has built up over the ages, but an inevitable consequence of the truth value of what we know. Karl Popper argues that one can never prove an hypothesis right and therefore we can never be certain we are right (Popper, 1963). However we can disprove hypotheses and consequently be certain that hypotheses are wrong. We can therefore progress in knowledge by disproving hypotheses. Popper has proposed this as a model for understanding the development of scientific knowledge. Others such as Kuhn (1969) have argued cogently that there are many other factors in why science changes direction. This is no doubt true, but the real value of Popper's hypothesis is not as a model for understanding the development of science, but for understanding what it means for anything to be "true". Truth lies in knowing that a hypothesis is wrong. It cannot be proven to be right, it can only be falsified. One can then reasonably ask: What is knowledge for? What is its relation to the world, particularly in terms of saying that something is true? Does it mean anything to say that we progress to a "truer" position as

†Note: Some sections of this paper are also included in "Towards hypothesis testing: JUSTIN, a prototype system using justification in context" by the same authors and submitted to the Australian Joint Artificial Intelligence Conference 1989

we disprove hypotheses. A major part of this paper will be an attempt to come to grips with such questions, because their answers will determine our analysis of what the proper task for knowledge-based systems should be.

For someone committed to building an expert system, these questions are not relevant. They become relevant only when one asks whether expert systems are the best way in which knowledge-based technology can be used. For the expert system builder it is not the philosophy but its practical consequences that matter. That is, it is inevitable that the knowledge the expert provides can never be an explanation of why a conclusion is right, it can only be an explanation of why other conclusions are wrong. In showing their chosen conclusion is preferable to other possible conclusions the expert does not show that their conclusion is preferable in any absolute sense, but only that it is less worse. And of course all these arguments are presented in a specific context, which determines which other hypotheses are rejected. The expert cannot disprove all hypotheses, only the limited set under consideration in the context.

Our previous work was directed towards methods of building and maintaining expert systems which attempted to recognise these aspects of knowledge (Compton and Jansen 1988 & 1989; Jansen and Compton 1988 & 1989). However the philosophical position we came to through this work suggested that although expert systems have been highly successful they are not the appropriate way to use knowledge which is justification in context. Before suggesting what we believe is a proper use of knowledge-based systems we need to develop further this philosophy of knowledge.

We have suggested that the hypothetical nature of knowledge applies not just to expertise but to all knowledge (Compton and Jansen 1989). Popper himself discussed childhood learning in terms of the same model and his ideas seem to fit well with Piaget's observations of early childhood development.(Piaget, 1929) Even so called "sense data" must be viewed as hypothetical since what we take as sense data is intellectually processed before we "sense" it. The "Logic of Perception" by Rock (1980) is devoted to a formal presentation of argument for this position. The most attractive example of this phenomenon is Ames trapezoidal room (Ames, 1951, described by Bateson, 1979). A small model of a trapezoidal room is constructed which when viewed through correcting prisms is seen as a normal rectangular shaped room. If one then tries to touch various parts of the room with a rod, it is difficult because one has a misleading view of the room. However after some time one can learn to touch the various parts of the room. The interesting result of the experiment however is that one then sees the room in its true trapezoidal shape, despite the information reaching ones eyes of a rectangular room. Other examples such as the discovery of perspective, limited colour vision in some groups etc. (see Compton and Jansen 1989 for a brief discussion) all suggest that the way we "see" the world fits with our hypotheses about the world and are therefore part of these hypotheses rather than primitives out of which we construct our knowledge.

It follows that knowledge is entirely made up, it is a pure construct. This is an inevitable consequence of agreeing that sense data is intellectually processed before it is "sensed" and is part of our hypotheses about the world. Although knowledge is "made up", this does not mean it is arbitrary. We use it to make sense of reality and our interaction with reality. The ability to recognise that something makes sense, Lonergan has described as insight (Lonergan, 1959). Archimedes' "Eureka" is the classic example. We all experience the flash of insight, the recognition that things hang together and make sense. Insight is not reduced to logic because we have to be able to recognise that a given logic "makes sense". Although we can experience the flash of insight, it is normally more humdrum and is normally accompanied by considerable work to ensure that what we think we are seeing really does make sense (Lonergan, 1959). A major part of this "work" is applied to trying to reach consistency and coherence between different insights. We are not going to cry "Eureka" over some insight where we have constructed some new knowledge entirely unrelated to what we have created for the rest of reality. There would be no insight there, no "making sense". Even major paradigm shifts are directed towards pulling together a whole range of experience, scientific or otherwise. The theory or modification of theory that explains some isolated occurrence, is no

theory and no insight. It doesn't make sense of anything. What we are recognising in insight is of course the intelligibility of reality, that property or aspect of reality whereby we can make up knowledge which makes sense of it.

None of this is meant to replace conventional theories of how the brain or memory etc. work. Such theories (e.g. neural nets) are very useful, but they tend to describe intelligence in a scenario where the data is predefined and pre-available to the mind and the question is how the mind draws conclusions from these data. What we are emphasising here is the intentional and active nature of intelligence. There is no data that is "given", we seek out what we need to make sense of things. The data that becomes available to the mind depends on what the mind chooses to collect what the mind "lets" the eye see (Rock, 1980). The mind controls not only what is paid attention to but also to what is "seen", e.g. Ames' trapezoidal room above and Sratton's well known inverting glasses (Stratton, 1897). These glasses inverted one's view of reality but after a while the world was "seen" right way up despite the glasses. Where then is the primitive data prior to intelligence? One cannot take the consensus reality of late 20th century man (to use Lenat's phrase (Lenat, 1989)) as some absolute given about which we reason. Consensus reality itself is a construct of our intelligence to make sense of our experience of being. This means that theories of how the brain works, often starting with "the brain *must* ...." or "neurons *must*....." themselves contain an hypothesis that brains are a suitable way of understanding parts of our experience. The brain and everything else is not an empirical given but an hypothesis like any other scientific hypothesis, which can be disproved and improved upon.

None of this leads to idealism - on the contrary. Insight is the perception that something does make sense; some knowledge structure that has been created works in enabling me to order and understand my interaction with existence. Because knowledge is created in a context it has to be continually reworked to ensure that it is based on insight, making sense of experience. The degradation of philosophies, cultures and religions when they lose touch with their grounding insights is obvious. The process of insight must be revisited as the context changes. This implies far greater realism than reducing intelligence to logical manipulation of some sort of Platonic primitives. It is interesting that the name "fundamentalism" is used to categorise religions and cultures that seem to have lost their grounding insights based in lived experience, and which attempt to conclude everything by logical deduction from primitive elements such as so called "literal" interpretation of texts. These movements have much in common with our hopes of building AI systems based on logical deduction from supposed primitives. We build expert systems as if heuristic knowledge has an immutable archetypal status despite the fact that it was acquired in a context and depends on that context. These statements may seem extreme but they are emphasised to counter the Platonism latent in much of so called commonsense (our current consensus hypothesis). True intelligence is an active process not concerned just with deductive reasoning but constantly checking that knowledge makes sense of reality and thus providing a basis for genuine realism.

Nor does this type of theory suggest solipsism or relativism. Part of the experience of reality of which we try to make sense is our experience of others and their statements in which they attempt to make sense of reality. We can ignore others as we often do, but if insight attempts to make sense of reality, then others and their attempts to express insight are a major part of the reality in which we exist. Again how we understand our experience, including our experience of others, is expressed in hypotheses. Our hypotheses of who and what others are and how we should relate to them are often very faulty, but hopefully we are prepared to seek new insights. In the Lonergan tradition, insight unrelated to the attempts of others to seek and express insight will result in very poor hypotheses. The position that is being proposed here is very much a realist position, but it is not an empiricist position where the world is a given and simply is as it seems. The realism that is proposed here depends on working, seeking new insight, seeking to stay in touch with reality. Without sufficient care people can drift off into unreality, logically following the implications of an hypothesis rather than seeking insight. Haers and Ydewalle (1989) in a study of knowledge acquisition based on verbalisation, note that retrospective verbalisations are more unreliable than verbalisation while the task is being carried out. This is not a question just of memory; when you are doing the task insight is still

active, giving the opportunity to check the knowledge being created. Shaw in her comparative study of different experts noted not only that experts used different language and seemed to structure their knowledge in different ways, but that they were used to dealing with these differences and did not necessarily make their disagreement explicit (Shaw, 1989). This seems to make sense only if one views knowledge not as an entity in its own right, as in the Platonic model, but as something that is put together to make sense of reality. If the experts are concerned with practical decisions it does not really bother them that they differ amongst themselves as to how to best organise hypotheses to make sense of reality. The experts would no doubt be better off with common knowledge structures, but their ability to communicate, to understand each other as experts and to function as experts, is not determined by the knowledge structure alone. Rather the knowledge structure and communication both depend on insight into reality.

Little reference has been made in this analysis to the ideas of Winograd and Flores (1987) and others who have also attempted to identify appropriate tasks for computers on one hand and human intelligence on the other. Much of their analysis seems consistent with what has been presented here, but with a different emphasis. These workers have primarily emphasised how intelligence is grounded in concrete experience. We have emphasised that although intelligence is grounded in concrete experience, the knowledge used to express the insights of intelligence is a construct, it is entirely made up.

KNOWLEDGE CONSISTENCY CHECKING: A COMPUTATIONAL TASK

What then is the role for computers in this, what are the possibilities for artificial intelligence and in particular for knowledge bases? Knowledge bases have generally been used as oracles. Feed in the data and out will pop the answer. We are well familiar with the brittleness of such systems. The explanation is not because they do not have enough commonsense as Lenat and Feigenbaum (1988) would suggest, but because they try to use an assemblage of justifications for conclusions given in particular contexts in place of insight. Although such systems are obviously useful and can find long term use (Buchanan, 1986) they have an inherent contradiction. They are meant to function as experts, but are always liable to act in a non-expert, perhaps completely ridiculous way, if used in a new and unknown context.

We have suggested that a key component in intelligence is checking apparent insight, checking if the apparent understanding fits into wider knowledge, other insights. This is not a new area in knowledge-based systems and tools have been developed to ensure that rules added to expert systems do not conflict or subsume existing rules(Suwa, Scott and Shortliffe, 1984). The difference here is that we propose that such tools have an important role in their own right in assisting the development of knowledge, not just expert systems. However other types of consistency checking other than subsumption etc. which are appropriate to production rules will be required depending on the type of knowledge being constructed. We judge that developing the right sort of consistency checking for different domains will be the major research problem for this application of knowledge-based technology; (we describe below our initial attempts to identify the appropriate approach for a medical research domain). The important point is that whereas interpretation of reality cannot be left to knowledge alone, but must be backed up by insight, the rules for finding inconsistencies in knowledge are themselves part of knowledge and so the problem is tractable for a knowledge-based system. The rules by which we construct hypotheses and whereby we identify inconsistencies are themselves part of knowledge, albeit at a higher level of abstraction.

While we propose that knowledge-based systems can be used to check knowledge for inconsistencies, they cannot be used to produce more coherent knowledge except in a trivial sense. To produce a more coherent knowledge depends on new insight. A new hypothesis within the existing framework may produce more coherent knowledge, or a paradigm shift with a whole new knowledge structure may have to be created. Thus knowledge consistency is a computational task, whether carried out by human or machine, but coherence depends on new insight. In the use of knowledge-based technology we are proposing, computers would check knowledge consistency and identify for the human knowledge builder, the need for new insight, a human responsibility.

Data has a special, that is, completely ordinary role in consistency checking. When we use "data" in checking that knowledge is consistent we normally describe the process as hypothesis testing. To us the two phrases are synonymous because of the hypothetical nature of data that we have outlined. We have described data as being part of the hypothesis in perhaps radical terms but it is obvious when one considers any experimental science. In the neuroendocrine domain we will discuss below, when there is a conflict between data and a hypothesis, it may be the hypothesis that has to be changed but on the other hand the meaning of the data may need to be changed. The conflict between the data and the hypothesis may the occasion that it is realised there is error in the measuring technique. Or it may be realised that the method is detecting some substance not previously attended to. Or the assumption that the substance being measured is an indicator for some process is mistaken, and it is not in fact the right indicator. On the other hand the experimental protocol may be the issue; perhaps the protocol causes all sorts of other effects that the experimenter was not aware of which explain the conflict between the data and the hypothesis, etc. etc. etc. Anyone familiar with experimental research will know that all of this is absolutely standard, although perhaps more common in biology than physics. There is no way of predicting what will be used to explain a conflict between data and hypotheses. The most outstanding example of this was the debate following the Benveniste paper in Nature (Davenas et al., 1988) showing apparent effects of molecules at infinite dilution. The correspondence in Nature following this paper is an object lesson in human creativity in the variety of hypotheses put forward to explain the results without allowing an infinite dilution effect, but without accusing the authors of misconduct.

Despite the hypothetical nature of data there is obvious advantage in using the conventional distinction between data and hypotheses and we will use these terms in the conventional sense in the latter part of this paper. The distinction is based on difference between new and established hypotheses and how the researcher uses this difference. The data are ways of describing reality based on more established hypotheses which the researcher is not expecting to be falsified by the experiment, and which the researcher takes as "true". On the other hand the hypothesis is new, not yet established and incorporated into the body of accepted hypotheses like the meaning of the data. The expectation is that the hypothetical meaning of the data will not have to be changed because of conflicts with new hypotheses, but as we have pointed out above this is far from certain.

It is not novel to suggest the use of computers to test new hypotheses; mathematical modelling and fitting models to data seem to have been with us as long as computers. The difference here is that we see automated hypothesis testing not just as a useful tool for scientific research but as the proper use of knowledge-based technology, an appropriate pairing of men and machines.

Also in the expert system area it is not novel to suggest that computers can be used to check human hypotheses. The expert critiquing systems pioneered by Perry Miller (1986) can be viewed as insight checkers or hypothesis testers. A physician enters patient information as well as a proposed diagnosis or plan of management etc. The computer then raises questions that the physician may have overlooked. In other words is the physician's proposal consistent with the knowledge base? Our proposal differs from Miller's approach in that knowledge acquisition and the growth of knowledge are not of primary importance to Miller. His knowledge bases are developed presumably by conventional techniques and are presumably intended to be fairly static. In contrast because of the philosophical analysis on which our proposal is based the fundamental issue is the creation of new knowledge. We are interested in checking new hypotheses against existing knowledge, rather than checking a specific judgement against existing knowledge. Obviously the two approaches are very closely related and the arguments Miller marshals for using a critiquing approach seem consistent with the philosophical analysis we have proposed. He points out that there is often no right answer in medicine and that therefore the more important role for the knowledge base is to *explain and justify* how what it has concluded fits with other knowledge (see also Teach and Shortliffe (1981)).

We are now in a position to explain better why in argument we always justify why we are right and do so by implicitly rejecting other hypotheses. As we have already discussed insight is more primary than logic and so we cannot use logical argument to get to insight. All we can do logically on the level of knowledge is to argue about consistency of knowledge. We can't claim to show consistency because it is impossible to say that we have checked consistency with everything that should be checked. We can however show inconsistencies and each inconsistency shown stands alone independent of whatever else may be shown. Hence all we can do in arguing why we are right is to show the inconsistencies in other hypotheses. We of course focus on the other likely hypotheses in the context thereby attempting to justify that we are correct. In the process of course we may lead the other person to insight and we can include all sorts of information to this end. But insight does not flow automatically from such communication, the person must see for themselves that something makes sense. From this point of view we can understand how the quality of a teacher's teaching is not determined by his logic and knowledge alone. We can also understand how arguments often degenerate to proving the other person wrong; in a formal sense it is all we can do. On the other hand the ideal approach to argument is to recognise that the point of view of the both parties will contain some insight somewhere, but the knowledge used by each to express it will be inadequate, mean different things to different people. Because we all make up knowledge, when we communicate our communication is always overloaded with unintended meaning and to communicate, as in anything to do with insight and knowledge, takes care and effort.

To return to the main theme of this section; in summary we suggest that knowledge-based technology could find a proper role, one in which the technology is perfectly matched to the task it is expected to do, in checking whether new knowledge that is constructed is consistent with preexisting knowledge. If inconsistencies were found it would then be up to the human user of the system to come up with a more coherent set of hypotheses, whether this depends on simple changes or major paradigm shifts. We are well aware of our own inconsistencies, how we hold different and often conflicting models of the world in different mental compartments. But a domain where the problem is more immediately soluble is perhaps medical research and our initial studies have been directed towards the sub domain of neuroendocrinology, the study of brain hormone interactions.

MEDICAL RESEARCH PROBLEMS

There is obviously a huge knowledge explosion going on in medical and other research in our time. There are now at least 2000 medical scientific articles published per week in internationally recognised journals. It is obviously impossible to read or even scan all this material. Modern technology has been extraordinarily successful in producing experimental results, but it is obvious that important hypotheses and data as well as carefully devised experimental techniques may be overlooked and ignored because of the sheer mass of information available, resulting at least in wastage and replication. Modern science has been so successful probably because of its essentially reductionist strategy. But the other side of a reductionist strategy is a divergence of hypotheses with no real means to ensure that growth in knowledge is consistent, except the diligence of individual scientists.

One aid to integrating knowledge has been dial up access to computer data bases and medicine is probably one of the most advanced areas in this technology with on line access to MEDLINE available since 1971 (Adams and Taine 1964, Lindberg and Humphreys 1989). However such text based systems are not appropriate for checking knowledge consistency which really requires a knowledge base. The material that may conflict with the researcher's new hypothesis may not be referenced appropriately, the key words and headings under which the paper is indexed may not encompass the reasons why it is relevant to the researcher's hypothesis. There is considerable research into ways of improving or replacing MEDLINE. Most of this is aimed at enhancing text retrieval but it is likely to be of major importance to our knowledge base approach in the long term. For example the Uniform Medical Language project sponsored by the US National Library of Medicine (Lindberg and Humphreys 1989) will be of obvious importance to medical knowledge base builders.

The ideal form of electronic publishing would include features whereby a researcher could describe his own hypothesis of interest and this description would drive a data base search to find all the published data that confirmed or denied the hypothesis. Such a system would allow the retrieval of data which had been published but the importance of which had been overlooked by the original author. For example, as will be discussed below, it is common in neuroendocrine research, to publish a range of data, but to use only some of this to argue for one's own hypothesis of interest, and thereby be unaware of the potential importance of this data to other hypotheses. One should be able to build up a whole network of hypotheses to find if any part of the network is contradicted by other data. The availability of such a tool would not only transform searching electronic data bases, but would provide a basis for unifying hypotheses in experimental sciences such as medicine rather than the fragmented approach we have at present. As one added in hypotheses, one would build up a network of hypotheses that as a whole intersecting model could be shown to be consistent with or conflict with known data.

Such a system would not be used just for publishing but would find its primary purpose in providing an environment for individual researchers for their research activity. It would provide the means for the researcher to hand over the to computer the job it is best able to perform: rapidly checking large amounts of material against moderately simple rules. The researcher has to have the insight to discover good ways out of the conflicts discovered. The researcher would keep his own accumulated data as well as data and hypotheses from all the literature stored in the system's data base. New hypotheses would be tested against this accumulated knowledge to look for inconsistencies that the researcher had overlooked. Ideally the system should have access to all the research literature but this is obviously a long way off. It would be a sufficiently powerful demonstration of this type of technology if it could handle the scientist's normal workspace of perhaps a few hundred papers apart from his own data. Our aim is to produce such a system and we describe below an initial prototype.

We again note a partial distinction between hypotheses and data. Although the meaning of data is hypothetical and open to revision, it is less hypothetical, or there is less expectation that its meaning will have to be changed than the new hypothesis that it is used to argue for. Although the researcher will be interested in how his hypothesis conflicts with the hypotheses of other researchers, he will be more interested in how it conflicts with the data of others, in that the meaning of the data is something that is reasonably well accepted by the community of scientists and something that his new hypothesis should be consistent with. But again in trying to resolve conflicts and arrive at new insight the researcher may equally hypothesise a new way of understanding the data as reject his model.

As a test domain we have chosen neuroendocrinology, which studies the control by the brain of hormonal and related systems. In particular for this paper we have taken as a sample hypothesis a model or network of interrelated hypotheses intended to explain brain control of glucose metabolism (see Fig 6. Smythe, 1989). Neuroendocrinology is a particularly challenging domain as the complex hypotheses (Fig 6) necessary to explain data are far from certain. There is a variety of hypotheses in the literature and considerable debate over their validity. Also it is common in neuroendocrine papers that authors will provide results on a whole range of parameters (Smythe, Bradshaw and Vining 1983) but use only some of this data to argue for the specific hypothesis addressed by the paper. Neuroendocrine papers therefore might be expected to provide data to confirm or deny hypotheses that were not actually addressed by the paper. It is also a difficult area in which to obtain data with a wide variety of novel techniques used. This results in a high frequency with which the meaning of data will be reinterpreted by later researchers to fit with their hypotheses.

THEORIES

To be able to test hypotheses or check whether knowledge is consistent we need some sort of rules. These rules come from the way hypotheses are constructed. Addis points out that hypotheses or models are built within the framework of what he calls a theory (Addis, 1989). The theory itself is a hypothesis but at a higher level of abstraction. Following Addis usage we will use theory to describe the source of the rules for consistency checking. Theories are never

clear cut and easily identifiable because we use whatever tools are necessary to construct hypotheses. Following Peirce (1934) Addis describes a theory as "any set of statements that forms some coherent description of the world that can be used to 'render facts likely'".

One common theory type uses space and time. We obviously construct all sorts of hypotheses about the world in terms of space and time. Space and time are so entrenched and so apparently real that we have difficulty in realising that they are hypotheses about the world which are good for building other hypotheses (d'Espagnat, 1983). They express important insight into reality but they are not "true". It is only with modern physics that we have been able to realise that they are hypotheses to be improved on, with evidence coming both from relativity and from quantum mechanics, particularly the demonstration of "action at a distance" with the experimental verification of the Einstein-Podolsky-Rosen paradox (Herbert, 1986).

We now discuss a number of candidate theories for hypothesis testing in neuroendocrinology.

Mathematical modelling

Mathematical modelling has long been proposed as a way of describing and testing hypotheses which is gaining increasing importance in assessing the behavior of very complex systems which are very well understood in terms of their individual components, for example weather forecasting, testing aircraft design, nuclear physics etc. However in many domains too many assumptions have to be made to get the model to run. Every relationship in the system must have a precise mathematical expression, whereas a researcher may only be able to describe his hypothesis in qualitative terms such as "A stimulates B". As a result when a simulation is carried out it can be impossible to tell whether the behavior produced by the simulation is due to what one was hypothesising or the extra assumptions required by the methodology. Because of this, simulation has failed to deliver its promise in many areas, particularly medicine where it has made little impact on experimental design. (McIntosh and McIntosh 1980) Strategies such as Monte Carlo simulation allow one to use a range of uncertainty for the values of numerical constants, but still require the mathematical form of functional relationships to be specified. Other strategies such as finite level modelling attempted to break the values for parameters into ranges which were "interesting" (Gann, Cryer and Shoeffler, 1972). However the categorisation of the values is too empirical and seems to have been little used. Obviously mathematics with differential equations and calculus provides a powerful and successful way of building hypotheses, but it is pointless trying to patch the theory with all sorts of additions so it can be used to construct hypotheses actually built with quite different theories.

Qualitative modelling

Another type of theory identified more recently is naive physics; i.e. people are able to understand some of the same processes physics is interested in without using physics. There have been a few attempts at computer implementations of naive physics such as Qualitative Process theory (Forbus, 1984) and Qualitative Simulation (QSIM) (Kuipers, 1986). QSIM in particular provides a well founded and elegant simulation algorithm. Quantities are described in terms of ordinal relationships and functional relationships are described as monotonic increasing or decreasing, avoiding the problems of the assumptions required to specify functional relationships in conventional modelling. Simulation is able to produce the complete set of qualitative behaviors for a given model, so a hypothesis could be shown to be consistent with data by showing the data had membership in the set of behaviors simulated.

Our initial attempts at building a hypothesis tester for the neuroendocrine domain were based on QSIM. A Prolog implementation of the QSIM algorithm by Coiera (1989) was extended to deal with the initialisation and the graphical language we used for description of hypotheses (see below). We were eventually forced to abandon the approach because of the very large number of behaviors produced by even simple models. Extensions have been suggested to control the branching of states, and a number of these were implemented such as higher order derivative constraint generation (Chiu, 1988), aggregation (Kuipers and Chiu, 1987) and even some of our own domain specific heuristics, but these did not control the problem on the types of systems we were interested in. Other problems include the production of spurious

(non-real) behaviors which would have ultimately presented difficulties in its use for hypothesis testing.

We surmise that the underlying reason for these problems with QSIM, which do not arise in human mental qualitative modelling or naive physics, is that although QSIM provides for qualitative description, it uses an essentially mathematical approach to find qualitative solutions to the series of equations which describe the system. There is no causality in the description; a functional relationship provides no information on whether parameter A affects parameter B or vice versa. In contrast, in naive reasoning one seems to always use causal arguments, the direction in the functional relationship is quite explicit. This is particularly so in medical research where researchers seem to invariably use causal language; for example, "the insulin increased because the blood glucose increased because the subject ate some glucose". There is interest in trying to derive causality from differential equations (Iwasaki, 1988), but in our domain causality seems to be the primary system description.

Causality and conservation of mass

Fig 6 shows a graphic representation of the network of hypotheses we will be dealing with below. This is *in press* in the conventional biomedical literature (Smythe, 1989) and is a fairly conventional representation of a complex of neuroendocrine hypotheses, and although by one of the authors of this paper predates and is independent from the work described here. The representation is essentially a compartmental model (MacIntosh and MacIntosh, 1980) but does not follow this representation strictly. Compartmental models are based on conservation of mass. They allow flows between compartments; the amount of substance in a compartment does not change unless there is a flow of material to or from another compartment. Chemical changes are also allowed within the compartmental modelling theory. Sources and sinks, that is compartments about which nothing is known except that they supply or remove material may also be included. As well as compartments and fluxes of material, compartmental modelling uses causal connections by which flux may be controlled by other compartments and fluxes. In Fig 6 causality is used loosely in that compartments are causally linked so that a change in concentration of one substance may effect the level of another substance without any reference to the fluxes. This looser representation than strict compartmental modelling seems widely used (but not necessarily a graphic representation) in papers in the neuroendocrine area as well as fairly widely in biology. A flow is only included in the model if something is known about the flux of material, otherwise causality between compartments is used directly.

The basic principle seems to be that changes must be explained. If the level of something goes up or down, then there will be a causal connection, perhaps through a whole series of intermediates, to something else going up or down which can explains the change observed. On the other hand a flux of material may explain the change and the change in flux itself will be explained a change in a compartment to which the flux is causally linked. The changes that have to be explained in most studies in the neuroendocrine domain are differences in results between different experimental treatment. If after treatment A, there is an increase in hormone X, but there isn't a difference after treatment B then some difference between A and B must account for the difference in X. This is then the general principle on which we based our consistency checker. Any difference in results between two experiments must be ultimately linked back to the different treatments given in the two experiments. This is a very simple principle but at this stage it seems adequate for the data and papers we have considered.

THE PROTOTYPE ENVIRONMENT

The prototype environment to carry out consistency checking in the neuroendocrine domain has been developed consists of three modules:

1. QMOD a module for graphic representation of hypotheses
2. A method for describing experiments including the experimental procedures as well as the data from the experiment. This part of the prototype is still at an early stage of development in that there are no tools for entering data or for being able to change the hypothetical meaning of the data apart from a text editor. However a further prototype is underdevelopment to store this material in a data base but to provide facilities for making the changes that hypothesis testing may make necessary.

3. JUSTIN the hypothesis tester.

The current system has been implemented on a Macintosh II.

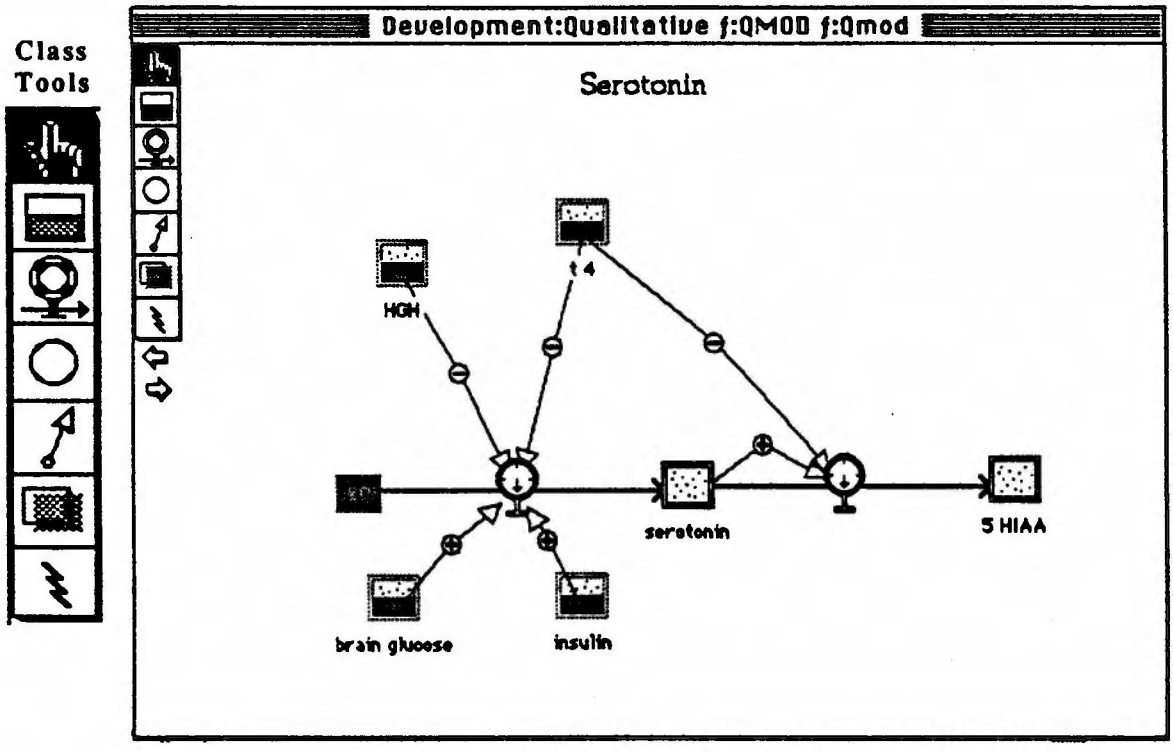


Fig 1. This shows the sub-hypothesis for serotonin regulation, as represented in the QMOD environment. The class tools, which are shown on the left, control the operations on a particular class of objects such as compartments. In this subsystem HGH is connected in a negative sense to serotonin production, which implies that an increase in HGH tends to decrease serotonin production.

QMOD : The Graphical Interface

It was assumed that the researcher using the environment would know little about computing, so an the interface was needed that was an intuitive extension of their existing computing knowledge. Hypercard™ was chosen to prototype the Builder. Hypercard is an object oriented framework which provides a good set of primitives to build a graphically based environment (Apple Technical Publication, 1987). QMOD stands for Qualitative MODelling. The builder allows researchers to maintain their hypotheses as a series of inter-related compartmental models. The creation and modification of these models is controlled by the tools on the left hand side of the card window (see Fig 1), and the pop-up object menus. The researcher simply draws the model on the screen. The tools follow the design of many Macintosh™ applications. Each tool controls the operations on a particular class of objects. The compartment tool controls the operations on compartments, so that to create a compartment the researcher selects the tool and then clicks on the card window. Objects can be moved by selecting and dragging, the links to other objects are automatically maintained. Clicking on an object reveals its class menu which allows operations on that class such as renaming or deleting. Deleting an object deletes all of its links as well. All the operations of the builder are accessible through this interface. The description of the hypothesis then becomes its visualisation within the builder's representation, providing a more direct and intuitive understanding of the system under study.

It can be seen in Fig 1 & Fig 2 that QMOD allows the construction of fairly conventional compartmental models, with compartments including sources and sinks, flows and controllers. Hypotheses are broken up into linked subhypotheses, each subhypothesis on a separate card. Compartments defined in a sub-hypothesis, such as serotonin in Fig 1, can be regulated only

from within that sub-hypothesis. hGH is represented in this sub-hypothesis as a shadow, because its regulation is defined in a different sub-hypothesis. Shadows are a separate class of objects and also have a class tool. The shadowing of a object involves copying and pasting the object into another sub-hypothesis and is as simple as the creation of any of the class instances.

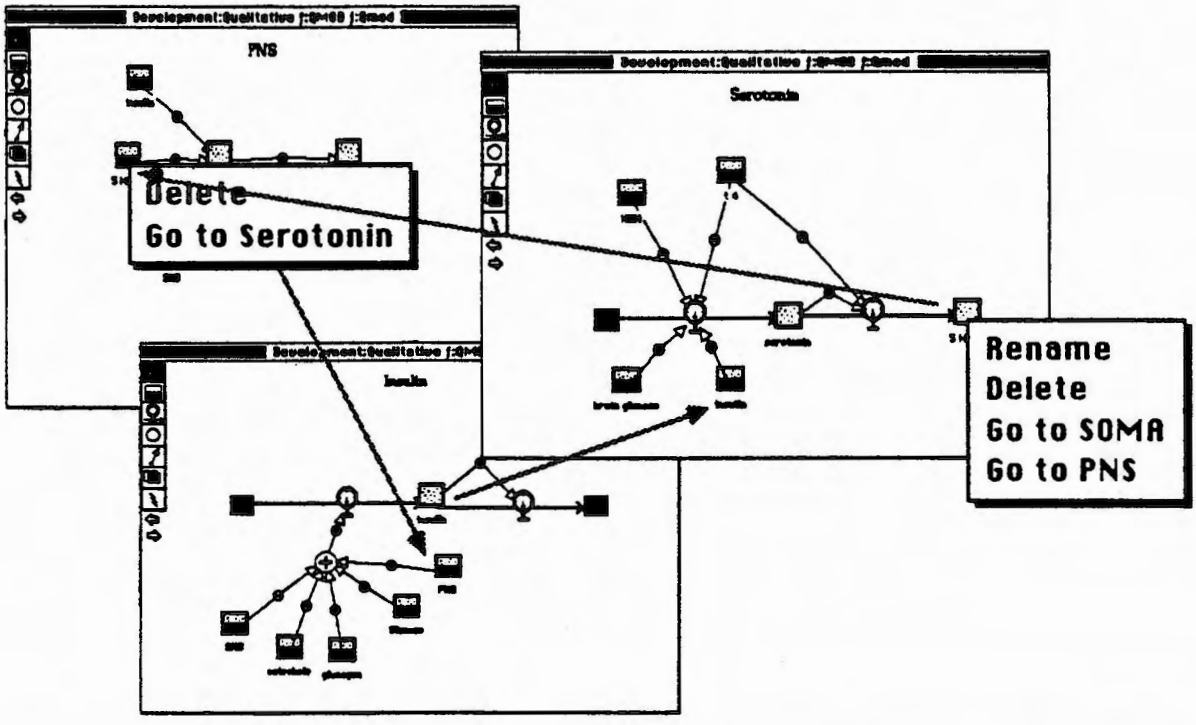


Fig 2. This shows the interrelations between different sub-hypotheses within the QMOD environment. Each model describes the regulation of a particular subsystem. The large arrows indicate the shadow relationships between different sub-subsystems which define the interrelationships within the central hypothesis. The bidirectional links between parent and child (shadow) are automatically maintained and accessible through the object menu (pop-up menus shown). From a shadow it is possible to navigate to its parent and from a parent it is possible to navigate to any of its children.

```
.....
compartment( [ serotonin ] ).
compartment( [ 5,hiaa ] ).
flow( [ serotonin ], [ 5,hiaa ], [ serotonin,to,5,hiaa ] ).
flow( somewhere, [ serotonin ], [ serotonin,production ] ).
connect( minus, [ hgh ], [ serotonin,production ] ).
connect( plus, [ brain,glucose ], [ serotonin,production ] ).
connect( plus, [ serotonin ], [ serotonin,to,5,hiaa ] ).
connect( plus, [ insulin ], [ serotonin,production ] ).
connect( minus, [ t,4 ], [ serotonin,production ] ).
connect( minus, [ t,4 ], [ serotonin,to,5,hiaa ] ).
.....
```

Fig 3. This shows the Prolog representation of hypotheses. This representation is automatically produced by the hypothesis builder; it is transferred to the tester where it can be compared against experimental results.

Fig 2 shows how a number of sub-hypotheses can be interrelated via shadow relations. As the hypothesis is built the bidirectional links between parent (real) and child (shadow) definitions are automatically maintained and are accessible through the object's menu. Navigating around the hypothesis is accomplished using the pop-up menus. Fig 2 shows the object menu for

5-hydroxyindoleacetic acid (5HIAA) and one of its shadows, by clicking on a shadow (child) it is possible to navigate back to its parent, whereas by clicking on a parent it is possible to navigate to any of its children. The total hypothesis is defined by the causal space that the shadow and causal relations span.

Using the environment it is possible to develop separate concurrent hypotheses and later merge them into a central hypothesis by describing their interaction. The hypothesis can be automatically translated into a Prolog representation (see Fig 3) and transferred to the hypothesis tester for comparison with experiments.

Experiments

The environment for describing experiments is at an earlier stage of development than QMOD, so that experiments are described in Prolog predicates. Experiments are represented as a set of experimental results and a description of the actual experiment in terms of the hypothesis. Each experiment has a label, as in *experiment([adrxstress])*. Fig 4 shows the description of this experiment. The results of the experiment are represented as a series of Prolog predicates such as, *result([adrxstress], [insulin] / 1366)*. In the result predicate, the first argument is the experiment label, and the second is the measurement. The measurement is represented as a name/value tuple, so that in the example above 'insulin' has the value '1366'. The values have been scaled to integer as the current version of Prolog does not support real arithmetic. This is not a problem, as the tester reasons only about ordinal relationships in the data, which are preserved during scaling.

The experiment is described as a set of actions. For example the experiment "adrxstress" involved both the removal of the adrenal gland (adrenalectomy) and stress. Each experimental action that is performed must be described in terms of the hypothesis. The lower part of Fig 4 shows a selection of the described actions in the test domain. These descriptions are represented as a set of conditions. For example the administration of the glucose analogue 2-deoxy-glucose(2DG) is used in the test domain to block brain glucose control of norepinephrine(NE) neurons. This is represented in the system as the statement,

condition([**2,dg**], disable(connect(minus, [brain, glucose], [ne, production]))).

which describes 2DG as disabling the causal connection between brain glucose and NE production. Adrenalectomy is described as two separate effects, one a decrease in catechole production, and two a decrease in cortisol production.

condition([**adrenalectomy**], state([catechole, production] / minus)).
condition([**adrenalectomy**], state([cortisol, production] / minus)).

As we have discussed such conditions are also part of the hypothesis and may be questioned, however we do not yet have the graphic tools equivalent to QMOD for manipulation these aspects of the overall hypothesis.

JUSTIN : Checking Experiments with Hypotheses

The approach to consistency checking that is appropriate for the domain is to check whether differences in experimental results between different experimental treatments can be explained in terms of the hypothesis and the hypothesised effect of the experimental actions. The hypothesis provides essentially a network of pathways between the experimental actions and the changes in the parameters, and for every change an appropriate causal pathway must exist. JUSTIN: **JUST**ification **IN** context, since the system can be viewed as justifying the hypothesis with the results or justifying the meaning of the results in terms of the hypotheses. The justifications are the pathways and the contexts the whole complex of hypothesis, results and experimental procedures. The prototype tester has been implemented in the UNSW Prolog version for the Macintosh, called Iprolog.

The system works by making comparisons between pairs of experiments in terms of the ordinal relationships between the experimental results

for example:

result([controlstress], [glucose] / 908).
result([adrxstress], [glucose] / 758).

The tester would derive justifications that explain glucose being greater in *controlstress* than in *adrxstress* in terms of the difference between the experiments *controlstress* and *adrxstress*. That is the justifications are based on the central hypothesis in the context of the difference between the experiments. So that when comparing an experiment involving adrenalectomy and stress to an experiment involving only adrenalectomy the tester produces justifications based on their difference, namely stress. Whereas when comparing adrenalectomy stress to control

```
. . . . .
. . . . .
experiment([adrxstress]).
result([adrxstress], [insulin] / 1366).
result([adrxstress], [glucose] / 758).
result([adrxstress], [dhpg] / 264).
result([adrxstress], [ne] / 1298).
action([adrxstress], [adrenalectomy]).
action([adrxstress], [stress]).

experiment([adrx]).
result([ adrx ], [ insulin ] / 684).
result([ adrx ], [ glucose ] / 614).
result([ adrx ], [ dhpg ] / 205).
result([ adrx ], [ ne ] / 1151).
action([adrx], [adrenalectomy]).

. . . . .
. . . . .
condition([adrenalectomy], state([catechole, production]/minus)).
condition([adrenalectomy], state([cortisol, production]/minus)).
condition([2,dg],
disable(connect(minus, [brain, glucose], [ne, production]))).
. . . . .
. . . . .
```

Fig 4. This shows the Prolog representation of experiments and conditions. Experiments are represented as a set of results, and a series of actions describing the experiment. The results are scaled to integer form (the current version of Prolog does not support real arithmetic). This is not a problem, because the tester reasons only about the ordinal relationships in the data. The actions taken in experiments are described in terms of the hypothesis as conditions. E.g. adrenalectomy is described as including a decrease in catechole production.

stress the justifications are based on adrenalectomy. The understanding of the system is developed through an analysis of the ordinal relationships between experiments. Each justification is composed of a series of causal events seen as a pathway through the hypothesis. In Fig 5. the causal justifications for glucose being greater in *adrxstress* than in *adrx* are listed and in Fig 6 these are shown diagrammatically over the domain model. The first in the listing states that glucose being greater in *adrxstress* than in *adrx* can be understood in terms of dhpg being greater, via the pathway:

dhpg/greater -> sns activity/greater -> liver glucose production/greater -> glucose/greater.

DHPG (the primary neuronal metabolite of NE) results from, and is therefore an indicator of, increased NE-neuronal activity which could be accompanied by an increased sympathetic nervous system activity, and which in turn could be accompanied by an increased liver glucose production, which could explain the increased glucose result. The justifications for a particular

```
resolving rel_result( [adrxstress], [adrx], [glucose], plus)

[dhpg] / plus -> [glucose] / plus
[
([dhpg] / plus              , relative_result([adrxstress], [adrx], [dhpg], plus)),
([sns] / plus               , connect(plus, [dhpg], [sns])),
([glucose_prod] / plus      , connect(plus, [sns], [glucose_prod])),
([from, liver] / plus       , connect(plus, [glucose_prod], [from, liver])),
([glucose] / plus           , flow(somewhere, [glucose], [from, liver]))]

[dhpg] / plus -> [glucose] / plus
[
([dhpg] / plus              , relative_result([adrxstress], [adrx], [dhpg], plus)),
([sns] / plus               , connect(plus, [dhpg], [sns])),
([glucagon, prod] / plus    , connect(plus, [sns], [glucagon, prod])),
([glucagon] / plus          , flow(somewhere, [glucagon], [glucagon, prod])),
([glucose_prod] / plus      , connect(plus, [glucagon], [glucose_prod])),
([from, liver] / plus       , connect(plus, [glucose_prod], [from, liver])),
([glucose] / plus           , flow(somewhere, [glucose], [from, liver]))]

[dhpg] / plus -> [glucose] / plus
[
([dhpg] / plus              , relative_result([adrxstress], [adrx], [dhpg], plus)),
([sns] / plus               , connect(plus, [dhpg], [sns])),
([pns] / minus              , connect(minus, [sns], [pns])),
([glucose_prod] / plus      , connect(minus, [pns], [glucose_prod])),
([from, liver] / plus       , connect(plus, [glucose_prod], [from, liver])),
([glucose] / plus           , flow(somewhere, [glucose], [from, liver]))]
. . . . . .
. . . . . .
```

Fig 5. Shows the representation used by the tester for justifications. The first justification links the change in glucose to a change in dhpg. The first step in this notes that there is a relative result of a difference in dhpg between adrxstress and adrx. The second step in the first relation states that the sympathetic nervous activity is greater in adrxstress than in adrx, because of the connection to dhpg etc.

result are all the pathways that could explain the change. When a result is greater the pathways that could explain that change are pathways that lead to either a decrease in inhibition or an increase in stimulation of the production of the substance. At the same time pathways that increase inhibition or decrease stimulation are possible, but cannot be used to explain this result. Justifications for a lower result are exactly the reverse, i. e. all pathways that lead to an increase in inhibition or a decrease in stimulation. Steady results are justified by comparing the set of greater result justifications to the set of lower result justifications, if one set is non empty then the other set must also be non empty.
It is possible that multiple justifications exist concurrently, Fig 5. shows a number of ways to understand the difference in glucose. The tester goes through each relative result for the experimental comparison and produces the total set of causal justifications. A justification is called **complete** if it starts with (is based on) one of the actions taken in the experiments, for example dhpg being greater can be understood in terms of the complete justification

stress/greater -> ne-neuronal-activity/greater -> ne-to-dhpg/greater -> dhpg/greater

where stress is an action of *adrxstress*. A justification is called **incomplete** if it starts with another relative result such as the justification above for glucose being greater. In resolving an experimental comparison all the results should be explained in terms of the actions, i.e. by complete justifications.

An experimental comparison is said to be resolved with the hypothesis if for each relative result there exists at least one complete justification.

To satisfy this from the basis of incomplete justifications is simple path traversal. In the examples above glucose being greater can be understood in terms of the complete justification:

stress/greater -> (dhpg/greater -> glucose/greater)

made up of the complete justification of DHPG and the incomplete justification of glucose in terms of DHPG. Consistency must also be maintained, in particular if no complete justifications exist for a relative result then all incomplete justifications that are based on it must be removed and the result labelled as unresolved with the hypothesis. The set of justifications must also be non-conflicting. Conflict may occur if experiments under-constrain the hypothesis producing multiple justifications. Thus one justification pathway may require factor X which has not be measure to increase, while another requires it to decrease. In practice this does not occur often because most experiments are designed to reduce ambiguity, so that the points at which conflict could arise are measured. In the event of a conflict the tester searches for a maximal conflict free set by binary splitting, and each relative result is labelled as either resolved, or unresolved with the hypothesis. If a relative result can not be resolved then either the hypothesis is inadequate or the meaning of the results is to be questioned. It is up to the researcher to use insight and come up with a solution to the conflict.

RESULTS

Our initial experiments have been aimed at checking whether the model described in Fig 6 is consistent with data in other neuroendocrine papers concerned with different but related hypotheses. We have used a set of 6 papers covering a range of hypotheses and all published from the Garvan Institute, so that we were in a position to go back to the original data if necessary; the third author on this paper was the principal author for all these papers.

The model in Fig 6 when translated into the QMOD formalism resulted in a hypothesis which included 27 compartments, 21 flows and 61 causal relationships spread across 14 sub hypotheses. The most interesting results were obtained from a paper on the effect of stress on corticotrophin (Smythe, Duncan, Bradshaw, Cai and Symons, 1983). This paper describes data on eight measured biochemical parameters, mostly brain monoamines. There were ten treatment groups, so potentially a very large number of comparisons could be made. The experimental animals were rats. The paper essentially argues that stress changes norepinephrine activity in the brain which in turn is responsible for the hormonal responses to stress. The paper also rejects the hypothesis that changes in serotonin turnover can be responsible for the stress response. The data and the assumptions as to the how the various treatments modified the model were entered as in Fig 4.

JUSTIN found some of the data in the paper to be inconsistent with model of brain control of glucose described in Fig 6 and taken from Smythe (1989).

a. the data for the norepinephrine metabolite could not be explained because the relation of stress to norepinephrine turnover had not been carefully enough translated from the rather vague representation in Fig 6 to the more precise QMOD representation. It was a trivial matter to change to a more appropriate description in QMOD of the norepinephrine subsystems.

b. Some of the serotonin turnover data could not be explained because we had failed to include in the QMOD representation the link from **pns** back to **serotonin** shown in Fig 6. This was a purely clerical error.

c. None of the dopamine metabolism data were explained by the model. This turned out not to be unexpected since the developer of Fig 6. and principal author of the paper in question was aware that he was unable to explain dopamine responses to stress.

d. The norepinephrine data in hypothyroid rats who had been given an α2 adrenegeric blocker could not be explained. This was a novel finding that the authors of the original paper were not aware of, and experiments are being planned to investigate further hypotheses which this finding has prompted.

Any of the results a,b or c would have provided validation of the method, as they found areas where the model conflicted with or was unable to explain experimental data. To find even these trivial results with any other simulation method would have been a very large task because of the size of the model. Of much more importance however is the finding of a physiologically interesting result. The reason for this result being found is that the authors had considered only the effect of the α2 adrenergic blocker on hypothyroid rats rather than the effect of hypothyroidism on α2 adrenergic blocker treated rats. That is they had considered treatment groups **A versus A+B,** but not **B versus A+B** although the paper contained data for both comparisons. Although the data is highly significant statistically, the comparison, one of hundreds possible in the paper, was not made because the authors were primarily interested in stress. They therefore tested the effect of stress in the presence of hypothyroidism, to see whether the same mechanisms were operative as in other stress situations. The reverse comparison looks rather at the effect of hypothyroidism in the presence of stress, a question the authors were not addressing. The result is of importance beyond understanding hypothyroidism since it suggests that the interaction between serotonin and norepinephrine described in the model will have to be relocated. This would be a major change in the model in Fig 6 and to our understanding of norepinephrine serotonin interactions. Experiments are being designed to provide more data on this question.

It is important to note that the developer of the model and principal author of the paper in question was not in any way involved in the test of the model against the data, and we had no expectation at all that this result would be uncovered. We suggest that the power of the JUSTIN approach that these results show is not because the algorithm is sophisticated. On the contrary the algorithm is extremely simple and appears to be the same sort of approach as humans use when justifying a link in an hypothesis or explaining experimental data. The only reason the computer program outperforms humans in this instance is the conventional one that computers are better at repetitive procedures that humans. In this case the human researchers failed to investigate all possible comparisons. We suggest therefore that the power of this approach will appear to increase as we build up to larger data bases of experimental findings and scientific papers.

DISCUSSION

To perform and make judgements in the real world relies on insight and therefore requires an insight-based system not a knowledge-based system. At present insight-based systems are no more than a research project (Addis, 1989) To use knowledge-based systems in real world decisions may be acceptable in some domains but has inherent problems in that there can be no guarantee that knowledge taken out of context will be appropriate. We have therefore argued that the appropriate task for knowledge-based systems is to assist in checking knowledge for inconsistencies.

In proposing that knowledge-based systems have a proper role in checking knowledge for inconsistencies we are not proposing some second rate role for the technology. One of the main features of our time is the largely uncontrolled knowledge explosion that has gone on, and we have made no progress at all in trying to integrate and pull together this knowledge. To achieve some progress in this area seems of far more importance than expert systems. It would provide an opportunity for a genuine blending of the respective strengths of men and machines if machines could take over the storage and consistency checking for knowledge, while man was responsible for generating new knowledge through insight.

Stepping back from the far frontier we suggest that the results above show that it is feasible to build such systems as personal tools. That we have found significant results in our first tests suggests that there is a need for such tools. We anticipate therefore that individual researchers

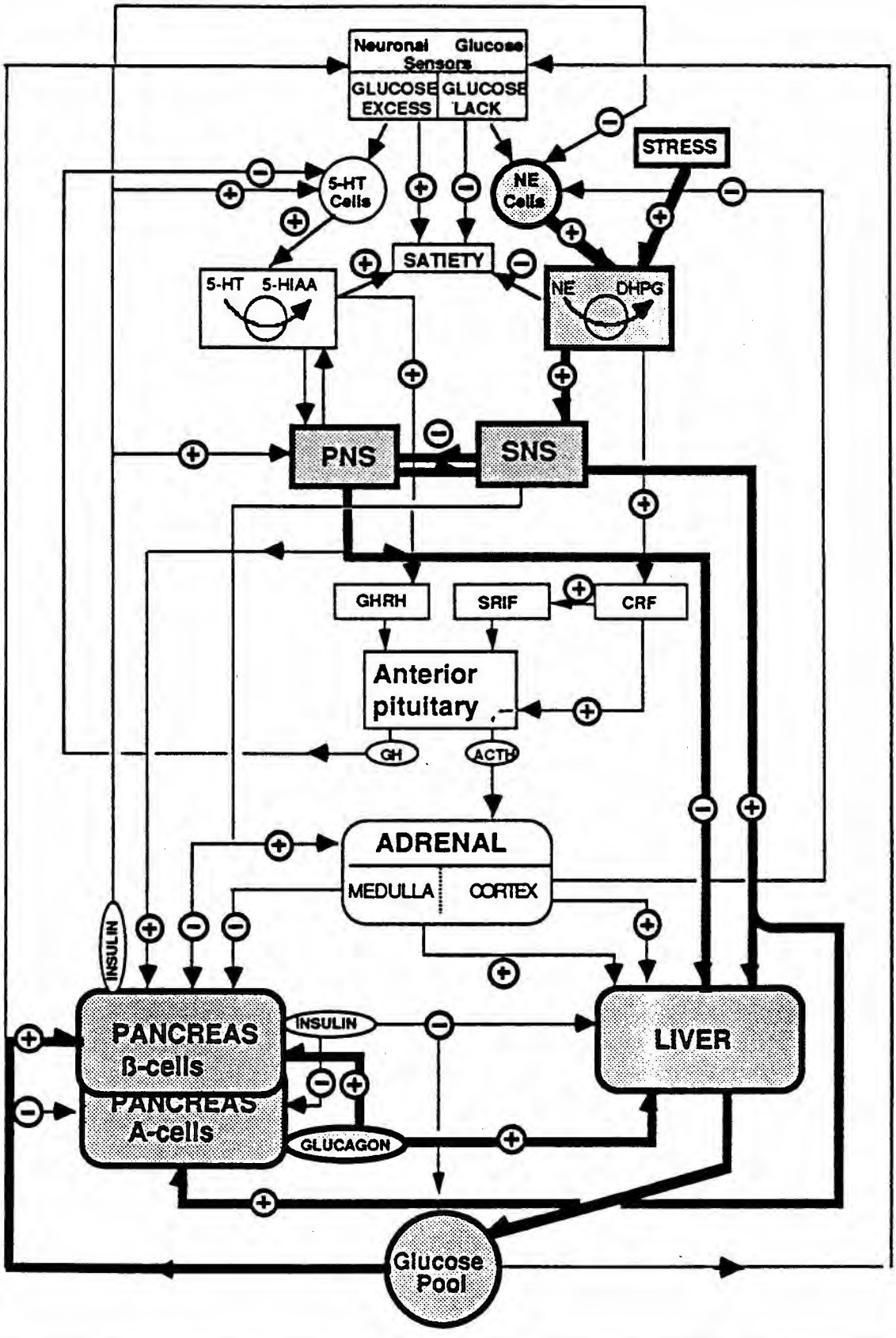


Fig 6. This shows the domain hypothesis as described in Smythe (1989). It is a model for brain control of glucoregulation. The pathways shown in bold are produced by the tester to explain the experimental comparison between 'adrxstress' and 'adrx' as described in the text.

will want to use such tools and be prepared to make the effort that their use will require.

The major research issue in developing this work further and applying it to other areas is in finding ways of checking for consistency, i.e. discovering theories that hypotheses can be built with. One obvious area where the work may be expanded is in including real number quantities rather than only ordinal relations. Although the JUSTIN approach is directed mainly towards structural or causal hypotheses we suspect that it may in fact go a long way towards handling quantities. JUSTIN only makes binary comparisons, but there is no limit to the number of comparisons that can be made. If there are a series of experiments which have a range of results for a certain parameter, JUSTIN will expect to find explanations in the model why a result is different from one experiment to another. The series of binary comparisons possible, should provide justifications or explanations for the range of results observed. Similarly the classification of "much greater" for results of one experiment compared to another, often is made because the researcher is aware of the results of other experiments that fall in between. If all this data is stored in the data base, the model should be able to explain the whole range of data. With expert systems "knowledge is power" and sophisticated reasoning strategies are not required. We suspect that with hypothesis testing, data is power and if all the data that is related to hypothesis formulation is stored, a simple algorithm like JUSTIN may be able to provide quite sophisticated testing of new hypotheses. However the domain for JUSTIN is still quite restricted and we anticipate significant research will be required to provide appropriate theories for the various domains.

A major requirement as we have intimated is for the researcher to be able to change any part of the information stored. The way the data is linked to the hypothesis, whether it is a measurement of one or maybe a family of substances, must be able to be changed. The supposed action of an experimental procedure in terms of the hypothesis may also be required to be changed. We propose initially to allow for this by simply allowing a series of meanings for data and experimental protocols; the original meaning plus that introduced by the researcher. Although this is not a large scale solution we estimate that it will be adequate in a tool for individual researchers. It will be important that any such changes be linked to a copy of the particular hypothesis that required their introduction. Similarly it would be reasonable in a complex developing hypothesis to link the different parts of it to the actual experiments that required the hypothesis to be generated. The description of the experiments and data will require full information on experimental preparation, measurement methodology etc. etc. We will also need some provision for statistical information; apparent differences may be due to error and possible differences may be masked by error. Again we need to be able to use not only the original author's assessment of the data but allow the researcher to override this with his own assessment so that appropriate ordinal relations will be calculated. Work on all these areas is underway based on a type of entity-relationship model (Jansen and Compton 1988 & 1989) but is still at an early stage.

Other developments include an expanded representation to include more structural as well as functional knowledge so that a richer description of the domain can be represented. Another problem which will have to be faced as the amount of data used becomes larger is the speed of the consistency checker.

ACKNOWLEDGMENTS

We are indebted to Claude Sammut of the University of New South Wales (UNSW) for making Iprolog for the Macintosh available to us and to Ric Coiera of UNSW who made his Prolog implementation of QSIM available and with whom we have had many profitable discussions on qualitative simulation. This work has been supported by an NHMRC project grant, now subsumed into an NHMRC Centre Grant to the Garvan Institute. The project to develop an environment for storing experimental data is a collaborative project with Monica Davis, the Biomedical Librarian of UNSW, and is funded by a UNSW General Development Grant. David Barda of the Garvan Institute is currently working on this project.

REFERENCES

Adams, S. Taine, S. (1964) Searching the medical literature *Journal of the American Medical Association* 188(3) pp.251-254

Addis,TR. (1989) Knowledge for design *Knowledge Systems Group, University of Reading Technical Report KSG/KTM/TR/10*

Ames, A Jr. (1951) Visual perception and the rotating trapezoidal window *Psychological Monographs* 65, whole no. 324.

Apple Technical Publications. (1987) *Hypercard Script Language Guide*, APDA.

Bateson, G. (1979) *Mind and Nature: a necessary unity* Wildwood House, London

Buchanan, B. (1986) Expert systems: working systems and the research literature *Expert Systems* 3(1) pp.32-51

Chui, C. (1988) *Higher Order Derivative Constraints and a QSIM-based Total Simulation Scheme* AI Lab. University of Texas Technical Report AITR88-65 .

Coiera, R. (1989) Intelligent Patient Monitoring *Proceedings of the Fifth Australian Conference on Applications of Expert Systems* pp.23-43

Compton, P. Horn, K. Quinlan, R. Lazarus, L. (1988) Maintaining an expert system *Proceedings of the Fourth Australian Conference on Applications of Expert Systems* pp.110-129

Compton, P. Jansen, R. (1988) Knowledge in context: a strategy for expert system maintenance *Proceedings of the Australian Joint Artificial Intelligence Conference (AI'88) Adelaide* pp.283-297

Compton, P. Jansen, R. (1989) A philosophical basis for knowledge acquisition *Proceedings of the Third European Knowledge Acquisition for knowledge-based Systems Workshop, Paris* pp.75-89

Davenas, E. Beauvais, F. Amara, J. Oberbaum, M. Robinson, B. Mindonna, A. Tedeschi, A. Pomeranz, B. Fortner, P. Belon, P. Sainte-Laudy, J. Poitevin, B. Benveniste, J. Human basophil degranulation triggered by very dilute antiserum against IgE *Nature* 333(6176) pp.816-818

d'Espagnat, B. (1983) *In search of reality* Springer-Verlag, New York

Forbus, K.(1984) Qualitative Process Theory *MIT AI TR 789*

Gann, DS. Cryer, GL. Schoeffler, JD. (1972) Finite level modelling of biological systems *Annals of Biomedical Engineering* 1 pp.385-445

Haers, M. d'Ydewalle,G. (1989) The problem of knolwedge extraction with human experts: a case of expert verbalizations *Proceedings of the Third European Knowledge Acquisition for knowledge-based Systems Workshop, Paris* pp.19-33

Herbert, N. (1986) How to be in two places at one time *New Scientist* 21 (August) pp.41-44

Iwasaki, Y. (1988) Causal Ordering in Mixed Structure *Proceedings of the American Association for Artificial Intelligence* pp.313-318

Jansen, R. Compton, P. (1988) The knowledge dictionary, a relational tool for the maintenance of expert systems *Proceedings of the International Conference on Fifth Generation Computer systems, Tokyo* pp.1159-1167

Jansen, R. Compton, P. (1989) The knowledge dictionary: storing different knowledge representations *Proceedings of the Third European Workshop on Knowledge Acquisition for Knolwedge-based Systems, Paris* pp. 449-461

Kuhn,TS. (1969) *The structure of scientific revolutions* University of Chicago Press, Chicago.

Kuipers, B. (1986) Qualitative Simulation *Artificial Intelligence* 29 pp.289-338

Kuipers, B. Chui, C. (1987) Taming Intractible Branching in Qualitative Simulation *Proceedings of the Joint International Conference on Artificial Intelligence, Milano.*

Lenat, DB. (1988) When will machines learn *Proceedings of the International Conference on Fifth Generation Computer systems, Tokyo* pp.1239-1245

Lenat, DB. Feigenbaum, EA. (1988) On the thresholds of Knowledge *Proceedings of the fourth Australian Conference on Applications of Expert Systems* pp.31-56 (Also MCC Technical Report AI-126-87)

Lindberg, DA. Humphreys, BL. (1989) Computer systems that understand medical meaning, in Scerrer,JR. Cote, RA. Mandil, SH.(Eds.) *Computerized natural language processing for knowledge engineering* Elsevier Amsterdam pp.5-17

Lonergan, BJ. (1958) *Insight* Darton, Longman and Todd, London

McIntosh, JEA. McIntosh, RP.(1980) *Mathematical modelling and computers in endocrinology* Springer-Verlag, Berlin

Miller, PL. (1986) *Expert Critiquing Systems* Springer-Verlag, New York

Peirce, CS. (1934) Scientific method, in Burks,AW. (Ed.) *Collected papers of C.S. Peirce*

Piaget, J. (1929) *The child's conception of the world* Routledge and Kegan Paul, London

Popper, KR. (1963) *Conjectures and refutations* Routledge and Kegan Paul, London

Rock, I. (1983) *The logic of perception* MIT Press, Cambridge, Mass

Teach, RL. Shortliffe, EH. (1981) An analysis of physician attitudes regarding computer-based clinical consultation systems *Computers and Biomedical Research* 14 pp.542-558

Shaw, MLG. (1988) Validation in a knowledge acquisition system with multiple experts *Proceedings of the International Conference on Fifth Generation Computer systems, Tokyo* pp.1259-1266

Smythe, GA. (1989) Brain-hypothalamus, pituitary and the endocrine pancreas, in Samols, E. (Ed.) *The endocrine pancreas* Raven Press, New York *in press.*

Smythe, GA. Bradshaw, JE. Vining, RF. (1983) Hypothalamic monoamine control of stress-induced adrenocorticotrophin release in the rat *Endocrinology* 113(3) pp.1062-1071.

Smythe, GA. Duncam, MW. Bradshaw, JE. Cai, WY. Symons, RG. (1982) Serotoninergic control of growth hormone secretion: hypothalamic dopamine, norepinephrine and serotonin levels and metabolism in three hyposomatropic rat models and in normal rats *Endocrinology* 110(2) pp.376-83.

Stratton, G. (1897) Vision without inversion of the retinal image *Psychological Reviews* 4 pp. 341-60, 463-81 (referred to by Kaufman L. (1979) *Perception:the world transformed* Oxford University Press, New York)

Suwa, M. Scott, AC. Shortliffe, EH. (1984) Completeness and consistency in a rule-based system In Buchanan, BG. and Shortliffe, EH.(Eds.) *Rule-based Expert Systems,* Addison Wesley, Reading, Mass. pp.159-70.

Waterman, DA. (1986) *A guide to expert systems* Addison Wesley, Reading, Mass.

Winograd, T. Flores, F. (1987) *Understanding computers and cognition: a new foundation for design*. Addison Wesley, Reading, Mass.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

AN APPROACH TO KNOWLEDGE ACQUISITION BASED ON THE STRUCTURE OF PERSONAL CONSTRUCT SYSTEMS

Kenneth M. Ford
Institute for Human and Machine Cognition
Division of Computer Science
University of West Florida
Pensacola, FL 32514

Jack R. Adams-Webber
Dept. of Psychology, Brock University

Frederick E. Petry
Dept. of Computer Science, Tulane University

Paul J. Chang
School of Medicine, Stanford University

Abstract

The purpose of this paper is to report on a continuing research effort aimed at the development and unification of the prerequisite underlying theoretical foundations for an adequate approach to knowledge elicitation from repertory grid data. A theory of confirmation is offered that incorporates the basic tenets of personal construct psychology directly into the logic as a basis for the determination of relevance, thus strengthening the logic, and extending personal construct psychology in an important and consistent manner. The results of this research have been applied to the development of *NICOD* – an automated medical knowledge acquisition system.

1. INTRODUCTION

Knowledge engineering involves the capture of human expertise in a computer system – thus permitting such expertise to be made widely available and hence useful to society in general. The development of expert systems has typically involved the knowledge engineer working closely with a specialist to acquire the relevant knowledge of the specialist's domain. This process has two significant drawbacks – it has been extremely laborious and domain experts often have difficulty articulating their knowledge in forms useful to the knowledge engineer. Indeed, Feigenbaum (1977) has suggested that knowledge acquisition is "the bottleneck problem of applied artificial intelligence."

However, the knowledge elicitation problem is one of long standing, in fact, Socrates may be considered the first knowledge engineer (Dreyfus and Dreyfus, 1986). In *The Euthyphro,* Plato relates a dialogue between Socrates and Euthyphro – an acknowledged expert on pious behavior (Plato, 1956). Socrates repeatedly asked Euthyphro about his rules or methods for ascertaining piety, and Euthyphro repeatedly responded with *examples* of pious behavior. Although Euthyphro was renown as a judge of piety, he was unable to articulate the rules or reasoning processes that he followed when making his judgments. Moreover, Socrates queried all manner of other experts, none of which could satisfactorily articulate their expertise. Thus, Socrates was no more successful than today's contemporary knowledge engineers.

Likewise, the problems of knowledge elicitation from a skilled person are well known in both clinical and experimental psychology. Psychologists have noted that there is no necessary correlation between verbal reports and behavior. Some researchers feel that problem solving knowledge derived exclusively from verbal data is nearly useless (Bainbridge, 1979). Several studies have observed that when domain-experts are asked to explain how they reach a given conclusion, they often construct plausible lines of reasoning having little or no relation to their actual problem solving methods (Johnson, 1983). Waterman (1986) has reported a particularly troublesome knowledge engineering paradox, that is, *the more competent domain experts become, the less able they are to describe the knowledge they use to solve problems.* A major reason that domain experts frequently find the verbalization of adequate decision rules daunting is that they apparently do not often structure their decision making in an explicitly formal way, and consequently, they may have considerable difficulty in isolating and describing their reasoning process (Shortliffe, 1976). Much of the knowledge possessed by experts consists of compiled experiential knowledge that is difficult to explicate. Improved knowledge engineering techniques (e.g., *NICOD* described below) endeavor to bring this sometimes subconscious knowledge to the surface and make explicit the heuristic knowledge that experts possess.

The availability of robust tools (i.e., shells) for building expert systems places an increased emphasis on the knowledge acquisition phase of the expert system development cycle. These increasingly powerful and popular software tools shift the burden of effort toward the necessary and difficult knowledge acquisition activities and away from programming activities. Such tools in conjunction with automated or semi-automated knowledge acquisition methods are facilitating the rapid development of prototype expert systems. Only through the development of effective knowledge acquisition tools will rapid prototyping become sufficiently inexpensive to permit alternative ideas to be tested and if necessary, "thrown away" (Boose, 1986). Several researchers have noted that the judicious use of a computer to interview subjects tends to alleviate many of the difficulties associated with the interview process (Gaines, 1981; Boose, 1984). Boose in particular has reported considerable success at Boeing in the computer automation of the knowledge elicitation process (Boose, 1985). The approach to semi-automated knowledge acquisition described in this paper is consistent with personal construct theory and has been employed in the service of knowledge engineers for the purpose of efficiently eliciting recalcitrant heuristic knowledge in the radiology domain (Ford and Chang, 1989).

1.1 Other Related Knowledge Acquisition Systems

In their efforts to loosen the knowledge acquisition bottleneck several researchers have successfully exploited personal construct theory's principle methodological tool – the repertory grid (see section 2.1). In this section we will briefly present the more notable related efforts in this area. ENTAIL (Gaines and Shaw, 1981, 1986) is an algorithm for the analysis of repertory grids which undertakes to derive logical entailments (dependency relations) between constructs, instead of statistical correlations. ETS (Boose, 1984) is a knowledge acquisition tool for both elicitation and analysis of repertory grid data. One of the several repertory grid analysis methods employed by ETS is based directly on ENTAIL. ETS has been used for several years at Boeing for the construction of prototype expert systems. AQUINAS (Boose and Bradshaw, 1987) is a flexible hybrid knowledge acquisition workbench incorporating several of the

analysis tools found in ETS (as well as others). KITTEN (Shaw and Gaines, 1989) is another hybrid knowledge engineering workbench that includes textual analysis, induction from behavior, provisions for multi-level and multi-expert repertory grid elicitation, and hierarchical construct laddering techniques. KSS0 (Gaines, 1987; Shaw and Gaines, 1987a) elicits knowledge via a repertory grid-based interviewing tool incorporating text analysis, behavior induction, and psychological scaling techniques.

2. PERSONAL CONSTRUCT THEORY

At the most fundamental level, the knowledge acquisition phase of expert system development consists primarily of eliciting knowledge from a skilled person. The literature of clinical and experimental psychology contains much work relevant to this attempt to improve communication between the knowledge engineer and the domain expert (Bainbridge, 1979; Hoffman, 1987). As noted earlier, several recent attempts to alleviate the knowledge acquisition bottleneck have been based either loosely or directly on personal construct psychology. Specifically, repertory grid analysis is becoming an increasingly useful tool in knowledge acquisition for expert systems (Boose, 1986; Ford, 1989). The root of the method's relative success at bridging the gulf between the semantic depth of the domain expert's knowledge and the limited nature of our entirely syntactic tools (i.e., computers) lies in the fact that the repertory grid is the major methodological component of personal construct theory – a complete and epistemologically sophisticated psychology. The judicious use of repertory grids may assist the knowledge engineer in avoiding the domain expert's cognitive defenses and thus elicit deep knowledge that the expert would not have been able to express otherwise.

Personal construct theory, as formulated by Kelly (1955, 1969, 1970) and summarized by Adams-Webber (1984), assumes that people typically use cognitive dimensions termed 'constructs' to evaluate their experience. Each construct, by definition, represents a single bipolar distinction. For example, a mother might use the construct 'excited/calm', among others, to interpret her children's behavior. Kelly argued that the underlying relation between the alternative poles of any construct is essentially one of bipolar opposition, or 'contrariety' (Husain, 1983). It follows that no construct can be understood fully without considering the meaning of both of its poles (cf. Adams-Webber, 1989).

Kelly (1955; p.68) posits also that *"Each construct is convenient for the anticipation of a finite range of events only."* It follows that each of a person's constructs has a specific *range of convenience,* which constitutes "all those things to which the user would find its application useful." Moreover, Kelly (1955; pp. 108-109) contends that any particular construct may have a somewhat different 'context' for each person who uses it, where its context comprises "the elements lying within the range of convenience of the construct (p. 108)." The range of convenience of a construct also limits its logical extension, since outside its range of convenience the construct is not applicable. When a person employs a particular construct to interpret events, for instance, she states that "David is outgoing", she is not only characterizing his behavior in a certain way (i.e., outgoing); but also contrasting it directly with behavior that she views as the opposite of outgoing, say reserved. It is likely, however, that she has observed people engaged in other actions that she cannot interpret usefully as either outgoing or reserved. Such conduct can be said to fall outside the range of convenience of the outgoing-reserved

distinction as she understands it. Nonetheless, it may lie within the range of convenience of other constructs in her personal repertoire, for example, wise-foolhardy.

From a Kellyan perspective, a necessary condition for organized thought is some degree of overlap between constructs in terms of their respective ranges of convenience (Adams-Webber, 1970). It is this overlap (or intersection) between the constructs' ranges of convenience that enables an event to be anticipated. His model of the *personal scientist* implies that each of us seeks to predict and control events by forming relevant hypotheses, and then testing them against available evidence. These hypotheses are derived from the relationships between constructs that articulate the 'logical' structure of an individual's personal construct system. More precisely, Kelly's (1955; p. 56) *organization corollary* asserts that "Each person characteristically evolves for his own convenience in anticipating events, a construction system embracing ordinal relationships between constructs". The research described below is concerned primarily with the development of a new method for eliciting the relationships between an individual's constructs on the basis of repertory grid data.

2.1 Repertory Grids

Kelly's Role Construct Repertory Grid, commonly called the 'repertory grid', is essentially a complex sorting test in which a list of elements are judged successively on the basis of a set of bipolar constructs (cf. Adams-Webber, 1987). The elements (e.g., personal acquaintances) can be either concrete or abstract entities, and may be said to operationally define the repertory grid's universe of discourse. These elements should be chosen carefully to represent the topic of interest as well as possible, and yet be of roughly the same type and level of complexity. We can either elicit a sample of 'personal constructs' from every subject individually, or simply supply the same 'standard' set of constructs to all subjects alike. Constructs represent the ways in which elements are judged to be similar or different from each other. In other words, they permit the subject to make relevant distinctions among the elements. Thus, a repertory grid may be considered a mapping of elements onto constructs. The data generated by each subject are entered into a separate two-dimensional table, or 'grid', in which there is a column for every element and a row for every construct. Each row-column intersect in this table contains a number indicating how a given construct was applied to a particular element. For example, the data represented in Figure 1 were elicited from a subject upon the evaluation of acquaintances.

Rate elements on a scale from 1 -> 5, a '1' is at the LHP, a '5' is at the RHP, while the rest fall between the poles.			Elements											
			1	2	3	4	5	6	7	8	9	10	11	12
C1	Somber	Humorous	2	4	3	4	4	4	2	4	3	4	1	5
C2	Individualistic	Group oriented	3	2	4	3	1	4	4	2	2	1	1	3
C3	Generous	Greedy	2	4	3	2	1	4	2	3	2	5	2	4
C4	Driven	Unmotivated	1	2	4	3	4	2	3	2	2	1	2	3
C5	Daring	Cautious	2	2	4	4	1	2	4	2	1	2	3	2
C6	Simple	Complex	4	3	2	3	4	4	4	2	4	4	4	2
C7	Honest	Dishonest	2	4	2	1	1	3	2	1	2	4	2	4
C8	Boistrous	Quiet	4	2	5	4	3	2	5	1	4	4	5	1

Figure 1 : An example of a multivalued repertory grid.

Kelly's repertory grid technique has played a crucial role in the development of personal construct theory, both as a research instrument and as a clinical tool (Neimeyer, 1985). In addition, this methodology is finding increasingly extensive application outside the explicit framework of personal construct theory; in fact, many investigators have used repertory grids in research having no logical relation to personal construct psychology (Adams-Webber, 1987).

During the last thirty years there has been a considerable proliferation of new forms of repertory grid tests in which people, objects, situations, or other kinds of elements, are either categorized, rated, or rank-ordered on a set of constructs. Several grid tests, unlike Kelly's original instrument, provide for the assignment of ratings on a n-point scale, with '1' being an assignment to the left-hand pole, and 'n' an assignment to the right-hand pole of a given construct (see Figure 1 above). The rating-numbers in the range (2 -> n-1) are employed to represent intermediate categories. To permit a natural midpoint, n is usually a small odd number, such as, 3, 5, or 7.

2.2 Approaches to Repertory Grid Analysis

The act of the respondent's assigning a rating to an element on a given construct has been interpreted in a variety of ways, and has thus lead to several different approaches to grid analysis, including information theoretic measures, non-parametric factor analysis, conventional factor analysis, principal component analysis, multidimensional scaling, and hierarchical cluster analysis, among others (Adams-Webber, 1987).

Kelly (1955) originally regarded the numbers in the rows of the grid as a vector of values representing the rating of each element in sequence on that construct. Several researchers have developed methods of repertory grid analysis that measure the distance between constructs in a multi-dimensional space. Some notable examples are, INGRID, Slater's (1977) principal component analysis, and FOCUS (Shaw, 1980), a distance-based cluster-analytic approach. PLANET (Shaw, 1982) is an integrated set of software tools for the interactive elicitation and analysis of repertory grids. These programs have been widely used in clinical psychology, as well as in the study of education and management.

Although extremely useful in some contexts, these distance-based approaches share the limitation that they can portray only symmetric relations. In other words, although they are able to detect that a given construct is similar to another construct, such methods cannot derive one-sided inferential relations (i.e., they cannot show that one construct logically entails another). Partly in response to this important limitation of conventional distance-based, cluster-analytic approaches, some researchers have developed procedures for the logical analysis of construct entailment (Gaines and Shaw, 1980; Ford, 1987). In such approaches, the grid data are viewed not as a set of vectors, but instead as an assignment of truth values to logical predicates. The left and right-hand poles of each construct are considered logical predicates that may be applied to the elements.

In binary rating systems, such as that utilized in Kelly's original repertory grid technique, the assignment of a '1' means that the predicate representing the left-hand pole of a given construct is 'true' for that element. Conversely, the assignment of a '0' indicates that the predicate representing the left-hand pole of a given construct is 'false', while the predicate representing the

corresponding right-hand pole is 'true'. Consequently, such 'binary' grids may be viewed as a matrix of boolean truth values. As a result, all entailments to be found in the repertory grid data may be derived through the application of a classical two-valued logic (Adams-Webber, 1979).

Although Gaines and Shaw (1980) found traditional two-valued logic adequate for the analysis of binary repertory grids, they propose a fuzzy semantics (based on fuzzy set theory) for the analysis of repertory grids that include a rating scale (Gaines and Shaw, 1981, 1986). Specifically, they regard each pole of a construct as a logical predicate that defines a fuzzy set, and the repertory grid as a set of such fuzzy sets. These fuzzy sets are then used for the derivation of entailments between constructs based on the Lukasiewicz L_1 form of fuzzy logic.

3. A LOGIC OF CONFIRMATION

The process of deriving construct relationships from repertory grid data can be related to the problem of induction. Induction may be considered the production of universal rules produced by generalizations based on a finite number of evidences. If the premises are true, it is merely probable that the conclusion is also true. Thus the inductive probability of an argument is a measure of the strength of the evidence that the premises provide for the conclusion. Fundamental to inductive logic is a related concept known as the logic of confirmation. Such a logic consists of the explicit formulation of rules for deciding whether or not an hypothesis or generalization is confirmed (i.e., supported by or lent credence) by some particular evidence, and if so, to what extent (Carnap, 1962; Hempel, 1965; Swinburne, 1973).

3.1 A Non-Quantitative Theory of Confirmation

The development of an appropriate non-quantitative logic of confirmation was a necessary condition for the subsequent elaboration of an adequate probabilistic interpretation of quantitative confirmation. The central problem in the study of the logic of confirmation has long been the problem of *relevance*. Elsewhere (Ford, 1987), we examined the paradox of the raven (Hempel, 1965) and found it disturbing that the existence of a white handkerchief could be formally shown to confirm the hypothesis 'all ravens are black'. This result offends our intuitions which hold the existence of a white handkerchief irrelevant to a hypothesis about ravens. The approach to non-quantitative confirmation adopted in this paper justifies this natural intuition.

Despite their formal equivalence in terms of symbolic logic, the hypotheses that 'all ravens are black' and 'all non-black things are non-ravens' are pragmatically very different. That is, they are not confirmed and disconfirmed by the same evidence. Von Wright (1966) noted that generalizations have an associated *range of relevance*, and consequently, only things within a generalization's range of relevance may constitute confirming or disconfirming evidence. All other things (things outside the range of relevance) are irrelevant. Furthermore, when a generalization's range of relevance is not specified (as is nearly always the case), it is taken to be the *natural* range of relevance, which is defined as *the class of things that fall under the antecedent term.*

Nonetheless, a serious practical problem remains. How is this 'natural range of relevance' to be determined? The class of items deemed as falling under the antecedent term will vary from person to person and over time. Thus, we contend that the problem of confirmation is at the

most primitive level, fundamentally psychological in nature, and will not submit to a purely syntactic approach. Kelly's (1955) personal construct psychology provides a comprehensive and systematic theory of human cognition, and thus seems particularly well-suited to the task of providing a foundation for a new approach to confirmation. Kelly's personal construct psychology is able to lend Von Wright's vague and theoretical concept of relevance a basis for realization. As noted above, in terms of personal construct theory, hypotheses are formed by the overlap or intersection of the constructs' ranges of convenience. Moreover, the repertory grid provides a method of determining Von Wright's range of relevance for a given hypothesis in the universe defined by the grid elements. Specifically, a theory of non-quantitative confirmation described in earlier work (Ford, 1987; Ford and Chang, 1989) incorporates the fundamental tenets of personal construct psychology directly into the logic as a foundation for the determination of relevance, thus strengthening the logic, and extending personal construct psychology in an important and consistent manner.

3.2 Epistemic Probability

Swinburne (1973) identified three common types or uses of probability:

- Physical: A measure of the extent to which an event is determined by causes;
- Epistemic: A numerical measure grading degree of rational belief in a statement, proposition, or hypothesis;
- Statistical: Propositions about proportions or relative frequencies which are often employed for the purpose of calculating epistemic probabilities.

In this work, we are primarily concerned with epistemic probability. The epistemic probability of a statement or hypothesis depends on the contents of the presently available supply of relevant knowledge. As a result, the epistemic probability of a hypothesis can differ from person to person and also vary over time.

For example, given the same item of radiological evidence (perhaps an x-ray), the epistemic probability assigned to the hypothesis "the patient is suffering from disease χ" is likely to be different for a first year medical student and an expert radiologist (see section 5). Regardless of their differing opinions, an absolute statistical probability that the patient is afflicted with disease χ may theoretically exist. This statistical probability, however, is unlikely to be available to a physician since the production of such an absolute statistical probability requires complete knowledge of all relevant cases. Presumably, the expert possesses some special stock of relevant personal knowledge that permits him to arrive at epistemic probabilities that more closely approach the absolute statistical probability than those offered by the novice. Future medical developments may prove our expert wrong, but such developments will be unable to show that his epistemic judgment was not probable on the evidence available at the time. It is these epistemic probabilities that guide our actions and serve as a basis for much of what we characterize as human expertise.

3.3 A Quantitative Theory of Confirmation

This non-quantitative theory of confirmation provides a basis for the development of a *quantitative logic of confirmation* consistent with personal construct theory. Traditionally there has been a great chasm between considerations of deductive and inductive logic. Inductive

logic is frequently regarded as a contradiction in terms, or at best as a poor sibling of deductive logic. However, inductive and deductive logic can be shown to be intimately related. Deductive logic is characterized by the concept of logical entailment, while inductive logic can be described by a relation of partial entailment. In fact, in this limited sense, deductive reasoning may be considered a special case of inductive reasoning. This situation is portrayed diagrammatically below (Salmon, 1973).

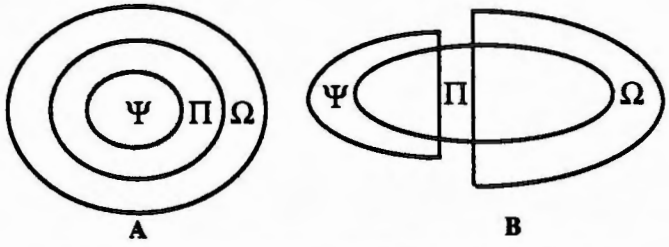


Figure 2: Pictorial representation of entailment.

In the figure above, the case of deductive logic is illustrated by situation A, which represents the universal propositions 'all Ψs are Π', 'all Πs are Ω', and 'all Ψs are Ω'. Situation B represents the case of inductive logic where the amount of overlap or partial entailment is measured by degree of confirmation. Note that situation B illustrates the propositions 'most Ψs are Π', 'most Πs are Ω', and 'no Ψs are Ω'. Thus, transitivity does not hold under this probabilistic interpretation of induction.

We consider a hypothesis such as, 'all Ψs are Π' to be a non-material conditional of the form, 'Whatever χ might be, if χ is a Ψ then χ is Π'. This conditional is thought of as affirming a bundle of individual conditionals (Quine, 1982): 'If χ_1 is Ψ, χ_1 is Π'; 'If χ_2 is Ψ, χ_2 is Π; and so on. The probability of such a proposition is based on a sample space of points in a universe w, corresponding to situations in which the proposition will be either true or false. In this manner, such non-material conditionals may be represented as bit strings consisting of 1s and 0s. For example, the symbol $i(\Psi)$ denotes the binary bit-string representing the occurrence or non-occurrence of Ψ;

if χ_1 is a Ψ then $i(\Psi)_1 = 1$,
if χ_2 is a Ψ then $i(\Psi)_2 = 1$,
if χ_3 is not a Ψ then $i(\Psi)_3 = 0$,
•
•
•
if χ_n is a Ψ then $i(\Psi)_n = 1$, otherwise $i(\Psi)_n = 0$.

Likewise, the symbol $i(\Pi)$ is referred to as the 'incidence of Π', and denotes an ordered subset of w consisting of all points in which Π occurs or is true.

Although diagrams such as those above are intuitively helpful, the task of measuring partial entailment poses severe difficulties. Any method founded on purely numerical mechanisms cannot provide the foundation for a probabilistic logic with truth-functional connectives (Bundy, 1985). Systems which assign numerical truth values to propositions or non-material conditionals are unable to provide a truth-functional probability logic without additional information concerning the relationships between the atomic components of the antecedent and consequent.

The bit strings or incidences described above provide the needed information for a truth-functional probability logic applicable to the measurement of partial entailment. Bundy (1985) has proposed an incidence calculus in which a set theoretic function is associated with each logical connective from propositional logic. This incidence calculus is now extended and applied to the measurement of partial entailment. These set theoretic functions are given below.

(i) $\imath(\mathbf{t}) = \omega$ (ii) $\imath(\mathbf{f}) = \{\ \}$ (iii) $\imath(\sim\Psi) = \omega - \imath(\Psi)$
(iv) $\imath(\Psi\ \&\ \Pi) = \imath(\Psi) \cap \imath(\Pi)$ (v) $\imath(\Psi \vee \Pi) = \imath(\Psi) \cup \imath(\Pi)$

Regarding our non-material conditional 'Whatever χ might be, if χ is a Ψ then χ is Π', there are four mutually exclusive and jointly exhaustive classes, which are given by the notation '$\{\chi: A\chi\}$' for describing the set of objects χ that have property A. The function $\mathbf{occ}(\imath(\Psi))$ returns the number of occurrences of Ψ in $\imath(\Psi)$.

$$P(\{\chi: \Psi\chi\ \&\ \Pi\chi\}) = \mathbf{occ}(\imath(\Psi) \cap \imath(\Pi)) / \mathbf{occ}(\imath(\omega)) = \mathcal{P}_1$$
$$P(\{\chi: \Psi\chi\ \&\ \sim\Pi\chi\}) = \mathbf{occ}(\imath(\Psi) \cap \imath(\sim\Pi)) / \mathbf{occ}(\imath(\omega)) = \mathcal{P}_2$$
$$P(\{\chi: \sim\Psi\chi\ \&\ \Pi\chi\}) = \mathbf{occ}(\imath(\sim\Psi) \cap \imath(\Pi)) / \mathbf{occ}(\imath(\omega)) = \mathcal{P}_3$$
$$P(\{\chi: \sim\Psi\chi\ \&\ \sim\Pi\chi\}) = \mathbf{occ}(\imath(\sim\Psi) \cap \imath(\sim\Pi)) / \mathbf{occ}(\imath(\omega)) = \mathcal{P}_4$$

and

$$\sum \mathcal{P}_i = 1,\quad P(\Psi) = \mathcal{P}_1 + \mathcal{P}_2,\quad P(\Pi) = \mathcal{P}_1 + \mathcal{P}_3$$

and thus

$$P(\Pi \mid \Psi) = \mathcal{P}_1 + (\mathcal{P}_1 + \mathcal{P}_2) = \mathbf{occ}(\imath(\Psi) \cap \imath(\Pi)) / \mathbf{occ}(\imath(\Psi))$$

To illustrate the use of these formulas, consider an example reflecting situation B, depicted in Figure 2. Let us assume the incidence bit-strings are as follows:

$$\imath(\Psi) = 0\,0\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0;\quad \imath(\Pi) = 1\,0\,1\,1\,0\,0\,0\,1\,1\,1\,0\,1;\quad \imath(\Omega) = 1\,1\,0\,1\,1\,1\,1\,1\,0\,1\,0\,1.$$

We are interested in measuring the degree of confirmation of the hypothesis 'All Ψs are Π'. The degree of confirmation associated with a hypothesis will fall in the range $0 \le C[h,e] \le 1$, depending on the amount of partial entailment given by a conditional epistemic probability as described below.

$$\begin{aligned} P(\Pi \mid \Psi) &= \mathbf{occ}(\imath(\Psi) \cap \imath(\Pi)) / \mathbf{occ}(\imath(\Psi)) \\ &= \mathbf{occ}(001000001010 \cap 101100011101) / \mathbf{occ}(001000001010) \\ &= \mathbf{occ}(001000001000) / \mathbf{occ}(001000001010) = 2/3 = 0.67 \end{aligned}$$

Under this interpretation the hypothesis 'All Ψs are Π' may be considered partially entailed to the extent 0.67.

4. RULE GENERATION

The ratings for a given construct in the repertory grid may be converted into a set A of measures of each element's position between the poles of the construct. The entire repertory grid may be considered a set of such measure sets $\{\mathbf{A}_1 \ldots \mathbf{A}_n\}$. These measure sets are a set of ordered pairs $\{(\chi, \mathbf{m}_A(\chi))\}$ where $\mathbf{m}_A$ represents a measure of the position of element χ on an interval valued line running between the poles of a given construct. If element χ_1 has a rating of r on construct c where the range of ratings is 1 through n, then χ_1 is positioned on the line between the poles of construct c at position $(n-r)/(n-1)$. For example, given the repertory grid in Figure 3, if $\mathbf{A}_1$ is taken to be the set of measures associated with the left-hand pole of the first construct, then the set of measures defined by the grid data are:

$$\mathbf{A}_1 = \{(\chi_1, 1.0), (\chi_2, 1.0), (\chi_3, .25), (\chi_4, 0.0), (\chi_5, .50), (\chi_6, .50), (\chi_7, 0.0), (\chi_8, .75), (\chi_9, .50), (\chi_{10}, 0.0)\}.$$

Clearly the range of the rating scale determines the various numerical values available for representing the position of an element on the interval valued line between the poles of the constructs. This leads to viewing repertory grid data as a collection of two-dimensional projections which we will refer to as α–planes. The repertory grid data above may be thought to consist of five α–planes, one for each possibility in the rating scale. Each α–plane is a matrix in which the elements can either be 1 or 0 according to the following definition:

$$\alpha\text{-plane}\,(c_i, \chi_j) = \begin{cases} 1 \text{ if } A_i(m_A(\chi_j)) = \alpha \\ 0 \text{ if } A_i(m_A(\chi_j)) \neq \alpha \end{cases}$$

We now present a repertory grid based on a manager-staff evaluation.

ELEMENTS 10, CONSTRUCTS 11, RANGE 1- 5
PURPOSE : Staff appraisal

	1	2	3	4	5	6	7	8	9	10	
1 intelligent	1	1	4	5	3	3	5	2	3	5	1 bonehead
2 introspective	1	1	5	4	2	3	1	3	4	4	2 superficial
3 beginner	1	2	3	5	4	4	4	1	4	3	3 experienced
4 motivated	1	1	4	5	2	2	5	3	3	2	4 unmotivated
5 reliable	3	2	2	5	1	1	5	1	2	3	5 undependable
6 mellow	3	4	5	2	2	3	1	5	4	5	6 abrasive
7 creative	1	1	5	5	2	3	4	3	4	5	7 not original
8 helpful	4	3	4	2	3	5	1	4	5	5	8 unhelpful
9 professional	1	2	3	3	2	1	5	2	4	4	9 unprofessional
10 sloppy	2	2	5	4	3	5	1	5	3	1	10 neat
11 overall superior	2	1	3	4	1	2	5	2	3	4	11 overall inferior
	1	2	3	4	5	6	7	8	9	10	

Figure 3 : A sample multi-valued repertory grid

For the sample repertory grid given above, the α–planes generated from the measure sets are as follows:

	1.0	.75	.50	.25	0.0
	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8 9 10
1 intelligent	1100000000	0000000100	0000110010	0010000000	0001001001
2 introspective	1100001000	0000100000	0000010100	0001000011	0010000000
3 beginner	1000000100	0100000000	0010000001	0000111010	0001000000
4 motivated	1100000000	0000110001	0000000110	0010000000	0001001000
5 reliable	0000110100	0110000010	1000000001	0000000000	0001001000
6 mellow	0000001000	0001100000	1000010000	0100000010	0010000101
7 creative	1100000000	0000100000	0000010100	0000001010	0011000001
8 helpful	0000001000	0001000000	0100100000	1010000100	0000010011
9 professional	1000010000	0100100100	0011000000	0000000011	0000001000
10 sloppy	0000001001	1100000000	0000100010	0001000000	0010010100
11 overall super.	0100100000	1000010100	0010000010	0001000001	0000001000

Figure 4 : Alpha-planes derived from the repertory grid data in Figure 3.

The 1.0-plane can be said to define all the "extremely" relationships in reference to the left-hand pole, while the 0.0-plane conveys all the "extremely" relations in reference to the right-hand pole.

Likewise, the .75-plane and .25-plane give all the "somewhat" relationships to the left and right-hand poles respectively, and the .50-plane indicates all the "neutral" relationships. In addition, the α–planes may be considered in various combinations. It is often desirable to produce rules that involve data found in more than one plane. For instance, a new composite α–plane may be produced by combining the 1.0-plane and the .75-plane to form a representation of "at least .75" or linguistically "at least somewhat".

We are interested in finding entailments between the constructs in various α–planes; these are not strictly logical or deductive entailments, but instead may be regarded as generalizations of the form 'Every A is B', where A and B are constructs. Given some heuristic rules governing the permissible combinations of planes, the set of all possible generalizations derivable from the repertory grid data will be called the hypothetical set ***G***. A generalization taken from ***G*** may then be evaluated by some rule/rules of confirmation to derive the degree of support provided by the evidence to the generalization.

Given the theory of confirmation described earlier (in Section 3) and our α–plane representation of repertory grid data, the following formalization of Nicod's Criterion (Nicod, 1970) is true:

For a given entailment $A \supset B$, or a hypothesis of the form 'Every A is B ',

let A represent construct c_i on $\alpha\text{-plane}_m$,
let B represent construct construct c_j on $\alpha\text{-plane}_n$, $i \neq j$,
let $p = \alpha\text{-plane}_m(c_i, \chi_k)$,
let $q = \alpha\text{-plane}_n(c_j, \chi_k)$,

a) An evidence is necessary and sufficient to confirm $A \supset B$ iff $\exists\, \chi_k$ such that $(p \wedge q)$ is true,

b) An evidence is necessary and sufficient to disconfirm $A \supset B$ iff $\exists\, \chi_k$ such that $(p \wedge \neg q)$ is true,

c) An evidence neither confirms nor disconfirms $A \supset B$ iff $\neg p$ is true.

In other words, adapting Nicod's Criterion to our terminology, it may be said:

a) The existence of an instance (an element) of a construct A on some α–plane which is at the same time an instance of construct B on some α–plane is necessary and sufficient to confirm a rule of the form 'Every (an instance of) A is (an instance of) B';

b) The existence of an instance (an element) of a construct A on a α–plane which is not at the same time an instance of construct B on some α–plane is necessary and sufficient to disconfirm the rule;

c) The existence of a thing (an element) which is neither an instance of construct A which is also an instance of construct B nor an instance of construct A which is not an instance of construct B neither confirms nor disconfirms the rule.

We can now illustrate these intuitively appealing judgments of confirmation and relevance by means of an example taken from the repertory grid in Figure 3. The modified relevance table (Sen, 1982) in Figure 5 provides a convenient means of illustrating the application of the proposed logic of confirmation to relevant examples. Consider a generalization taken from the set of ***G*** such as "Every '*extremely* introspective' is '*extremely* intelligent' ", which can be considered equivalent to the hypothesis that "*extremely* introspective entails *extremely* intelligent".

EXTREMELY (1.0) INTROSPECTIVE entails EXTREMELY (1.0) INTELLIGENT strength = .67

INTROSPE - INTELLIG		INTROSPE - $\overline{\text{INTELLIG}}$		$\overline{\text{INTROSPE}}$ - INTELLIG		$\overline{\text{INTROSPE}}$ - $\overline{\text{INTELLIG}}$	
evidence - 2	+	evidence - 1	-	evidence - 0		evidence - 7	
RELEVANT		RELEVANT		IRRELEVANT		IRRELEVANT	

Figure 5 : Relevance table demonstrating the application of the proposed method of rule generation to an example from the repertory grid in Figure 3.

As illustrated in Figure 5, the '+' in the small box indicates that the evidence in addition to being relevant, is also positive or confirmatory, while a '-' indicates negative or disconfirmatory evidence. A numerical measure of the degree of confirmation of an entailment may be had by dividing the confirmatory evidence by the total relevant evidence. In the example above, the entailment is confirmed by the evidence to a strength of .67.

Note that though this method is related to the standard conditional probability, it is not equivalent. The standard conditional probabilities produce a positive probability for the hypothesis *A* entails *B*, even when *A* and *B* are not related by any relevant evidence. If we are interested in evaluating the strength of the hypothesis *A* entails *B*, we have $p(B \mid A)$; and if there is no relation between *A* and *B* we have $p(B \mid A) = p(B)$, thus possibly producing a positive probability. From the repertory grid data in Figure 3, let *A* = 'extremely (1.0) intelligent' and *B* = 'extremely (1.0) reliable'; the application of standard conditional probability would produce a positive strength of entailment where no entailment exists.

EXTREMELY (1.0) INTELLIGENT entails EXTREMELY (1.0) RELIABLE strength = 0.0

INTELLIG - RELIABLE		INTELLIG - $\overline{\text{RELIABLE}}$		$\overline{\text{INTELLIG}}$ - RELIABLE		$\overline{\text{INTELLIG}}$ - $\overline{\text{RELIABLE}}$	
evidence - 0	+	evidence - 2	-	evidence - 3		evidence - 5	
RELEVANT		RELEVANT		IRRELEVANT		IRRELEVANT	

Figure 6 : Table demonstrating a hypothesis with no relevant confirming evidence.

This problem is caused by the use of irrelevant evidence in the calculation of the conditional probability. As illustrated by the relevance table in Figure 6, the only relevant evidence is disconfirmatory, which necessitates an entailment strength of zero.

4.1 A Comparison to the Fuzzy Logic Approach

The theoretical rationale of the fuzzy approach, as does that of any other approach to the analysis of repertory grid data, requires an explicit interpretation of the response of assigning a rating to an element for a given construct. In all methods derived from fuzzy set theory, assigning a rating is interpreted as providing a representation for that element's degree of membership in the fuzzy set defined by a pole of the construct (Boose, 1986; Shaw and Gaines, 1987b). Some researchers (Ford and Adams-Webber, 1989) have suggested that the underlying assumptions entailed by this interpretation frequently differ significantly from those held by the domain expert – thus possibly distorting the knowledge derived from the grid.

In Kelly's (1955) original repertory grid test, each element was assigned to one of two dichotomous categories for each construct. The shift from a two-valued rating scale to a multi-valued scale does not necessarily change the essential nature of the task itself. It may simply provide the subject with more degrees of freedom in responding, for example, five categories as opposed to two. In contrast to the assumptions underlying the fuzzy set approach, a repertory grid can be viewed as the record of the successive allocations of a list of elements to a set of scaled bipolar constructs. From this perspective, a subject can be seen as having made a 'crisp' (unambiguous) distinction concerning the placement of each element on an interval valued line containing the number of points used in the rating scale. It would follow that, the ratings as such are not 'fuzzy'. For example, when a five-valued scale is employed, each of the subject's judgments will fall onto a line segment as illustrated below in Figure 7a, while distinctions made on a seven valued scale can be mapped onto line segments similar to that depicted in Figure 7b.

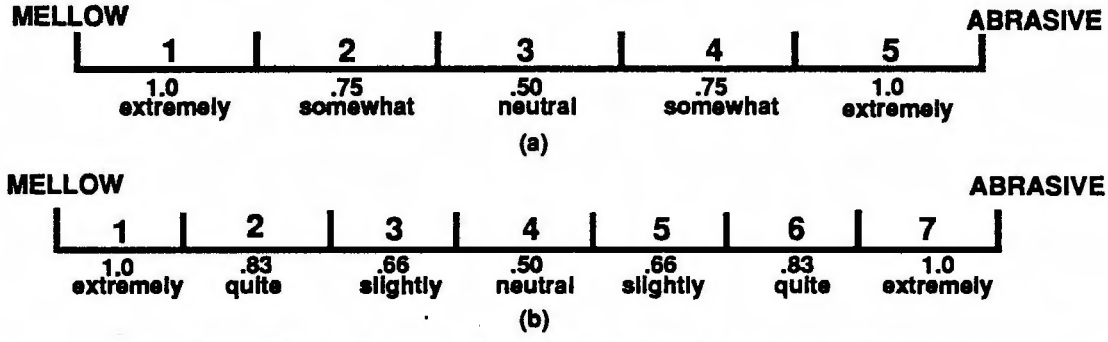


Figure 7 : An interpretation of multi-valued repertory grids

For many applications, the use of either five or seven-valued scales seems appropriate (Chang, Ford, and Petry, 1987). In fact, with respect to seven-valued ratings, researchers have found that when taken over a large number of subjects, all seven categories tend to be used with roughly equal frequency (Osgood, Suci, and Tannenbaum, 1957).

Previous approaches to knowledge elicitation from repertory grid data have generated entailments such as that illustrated below (Boose, 1986; Gaines and Shaw, 1986):

intelligent ⊃ creative Strength = (.75).

The approach described here will also produce such general entailments; however, under some circumstances it is interesting and useful to increase the magnification and generate hedged entailments such as:

extremely(1.0) intelligent ⊃ *extremely(1.0)* creative Strength = (1.0);
at least *somewhat(.75)* intelligent ⊃ at least *somewhat(.75)* creative Strength = (.40).

We find that reliance on the former approach may in some cases distort or obscure the expert's knowledge. For example, '*extremely* motivated' does not necessarily entail the same degree of 'overall superior' as '*somewhat* motivated'. Human experts are not limited to reasoning about broad concepts like 'intelligent'; but instead, they subdivide such concepts by degree and may make inferences based on these finer distinctions. In fact, this ability to make and use fine distinctions in reasoning is fundamental in our concept of an expert.

The addition of a hedge to a predicate is viewed here as effectively creating a new predicate. This is partly because of the fact that for hedges like *very* there is no generally applicable operation with which it can consistently and appropriately modify the original predicate. The meanings of hedges like *very* depend largely on the nature of the term and domain to which they are being applied. For example, in fuzzy set theory *very* is usually defined as a power operator. This somewhat ad hoc choice leads to many surprising results which are well documented in the literature (Haack, 1979; Mamdani and Efstathiou, 1984). Under some circumstances the *very* operator should be appropriately defined as a power operator (concentration), and in other circumstances, should be defined in some other way, perhaps as a shift operator (Hersh and Caramazza, 1976). Our proposed method does not suffer from these difficulties because hedges are not considered operators on fuzzy sets representing predicates.

The necessary distinction between "truth" and "precision" is illustrated by the following example. Consider the grid in Figure 3, subject-4 has a rating of 2 for construct 6. An interpretation based on fuzzy logic would produce (s1) below (or hold that s1 and s2 are equivalent) while we find that (s2) is the more appropriate interpretation:

" 'subject-4 is mellow' is somewhat(.75) true." (s1)
" 'subject-4 is somewhat(.75) mellow' is true." (s2).

The above statements are clearly not semantically equivalent. In (s1) *somewhat* is either semantically redundant or refers to the certainty or degree of truth associated with the proposition 'subject-4 is mellow', while the *somewhat* in (s2) refers to where subject-4 falls on the line running between "mellow" and "abrasive". Truth values applied to propositions derived from repertory grid data, such as (s2), should be considered *binary* (i.e., completely true or completely false). If subject-4 is a borderline case, the applicability of a category may be uncertain (until the user makes a distinction), but not 'middle true' (Schefe, 1981).

The fuzzy approach seems inappropriate on the grounds that fuzzy logical statements, such as (s1), are statements that are true to some numerical degree between 0 and 1 (see Hetherington, 1989). However, the repertory grid test has no provision for eliciting fuzzy or vague data from the user or expert. Instead, the meaning of the elicited repertory grid data is more consistent with an analysis based on a probabilistic logic of confirmation in which a hypothesis is assigned a numeric likelihood of being absolutely true. The assignment of a rating for an element on a given construct may be viewed as part of a *precisification* process; in other words, the question is not, "How true is it that he is mellow?", but rather, "How mellow is he?". Consequently, the *somewhat*(.75) rating is interpreted not as a matter of truth, but of precision.

Much of the research effort directed at knowledge elicitation from repertory gird data has tended to focus rather exclusively either on the act of eliciting the grid itself or on its subsequent analysis. As a result, insufficient attention has been paid to the *meaning* of the elicited data and the semantics of the expert's act of discrimination in which an integer value is assigned to the intersection of each construct and element. This lack of introspection has sometimes lead to the use of inappropriate logics in the production of expert system rules.

5. AN APPLICATION

This work is an integral part of a continuing medical knowledge acquisition project (Ford and Chang, 1989). We are particularly interested in developing an improved method (i.e., both in terms of efficiency and in the quality of the knowledge elicited) for the automated generation of expert system rules in radiology. For such a system to be successful, it must assist the knowledge engineer in bringing the expert's deep compiled knowledge to the surface, thus making explicit the valuable heuristic knowledge that experts possess but are frequently unable to articulate. It is also crucial that the interaction with the physician during the knowledge acquisition process be as "natural" and "non-intrusive" as possible; rather than force the radiologist to work within an awkward and "unfamiliar" environment, the process must "conform" to the radiologist and his/her domain.

We have found the spreadsheet paradigm a most efficient and agreeable grid elicitation technique for the radiology domain. The spreadsheet's visual metaphor amplifies the expert's ability to recognize and offer distinctions between the elements on the basis of the constructs. The conceptual model (spreadsheet) used to interact with the expert hides the complexity and unfamiliarity of the model (rules) upon which the knowledge base is actually structured. The flexible spreadsheet user-interface assists the domain expert in visualizing all (or many) of the repertory grid relations at once. Domain experts can quickly become proficient in the use of the spreadsheet interface, thereby reducing the expert's expensive and sometimes awkward reliance on the knowledge engineer. At any time during this elicitation process the expert may choose to appraise the rules produced thus far, and if necessary, return to the grid and make adjustments.

In addition, the expert may send a rule set directly to a visually oriented spreadsheet-based expert system simulation tool, thus permitting the expert to evaluate the functional performance of the rule set under development. By choosing the appropriate templates (from a set of predefined options) the user can cause the system to simulate the behavior of a variety of expert system programming paradigms (and shells) on the rule set in question. In this manner, very rapid mini-prototypes can be constructed. This interactive knowledge acquisition cycle is illustrated below in Figure 8.

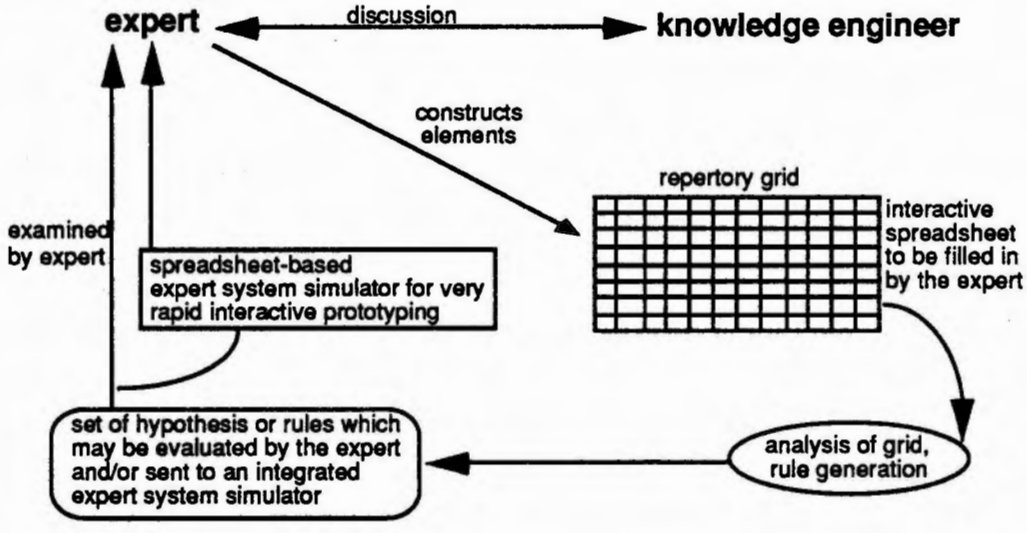


Figure 8 : A pictorial representation of the proposed knowledge acquisition process.

In application domains such as radiology, the elements frequently represent diseases or other pathologic states, while the constructs delineate symptoms (or other historical information), physical/laboratory data, or findings from a diagnostic imaging test. It is important to note that the elements are not cases or instances of a particular disease or condition (data), but rather represent the expert's concept of the disease or condition. The expert's ratings are formed on the basis of a summation of many years experience, and as such, represent knowledge as opposed to data. Consequently, the system described is intended to elicit expert knowledge, and represent it in terms of epistemic probabilities, rather than perform statistics on large case files.

The following repertory grid was produced by an university faculty radiologist with an expertise in mammography without undue difficulty and with relative independence from the knowledge engineer. This physician was impressed by how efficient and "natural" the knowledge acquisition process became when using the repertory grid approach. This should not be surprising, since the routine, everyday process of evaluating diagnostic imaging tests used by the radiologist involves identifying pertinent findings (elements) and characterizing them (assigning a rating within each construct). After this process is performed exhaustively, an impression or diagnosis offered. The striking similarity between this process and the repertory grid is obvious.

Element Index:

1 - - Fat Necrosis (post biopsy)
2 - - Fibroadenoma
3 - - Focal Ductal Hyperplasia
4 - - Focal Adenosis
5 - - Radial Scar
6 - - Focal Dysplasia
7 - - Cystosarcoma Phylloides
8 - - Fibroadenoma Degeneration

Predominantly benign lesions of the breast rated on a scale from 1 - 5. ('1' is at the LHP, a '5' is at the RHP)			Elements							
			1	2	3	4	5	6	7	8
C1	no biopsy order	certain biopsy	1	5	5	5	5	5	5	1
C2	low density center	hi density center	1	3	4	3	2	4	5	5
C3	diagnosis benign	diagnosis cancer	1	3	4	5	4	4	5	1
C4	diffuse	dominant	5	5	4	4	2	5	5	5
C5	nonpalpable	palpable	3	4	5	1	1	1	5	1
C6	contour smooth	stellate	4	2	4	5	5	4	3	2
C7	non char ca ++	char ca ++	1	1	3	3	1	1	1	2
C8	small size	large size	5	5	4	3	4	4	5	3
C9	ground glass	dense	2	2	3	3	3	2	5	2
C10	no ductal prolif.	ductal prolif.	1	1	5	1	2	3	1	5
C11	no dysplasia	fibrous dysplasia	1	5	2	4	1	3	4	1
C12	no distortion	neighbor distortion	2	1	4	4	3	1	1	1
C13	expansile	retractile	2	2	2	4	5	1	3	3
C14	no skin nodes	skin nodes	1	1	1	1	1	1	1	1
C15	symmetry	asymmetry	5	5	4	3	4	5	5	5

Figure 9 : A sample repertory grid from the radiology domain

The system quickly produced an interesting and useful set of rules. Several of these rules provide particularly suitable examples of major advantages associated with this approach to knowledge engineering; one such rule is:

(R1) IF *extremely* (1.0) confident in not ordering biopsy
THEN *extremely* (1.0) no dysplasia (1.0).

Considerable breast dysplasia (fibrocystic disease), although not necessarily dangerous, can obscure other more potentially significant lesions, thus making a confident interpretation difficult. This rule (R1) suggests that *extremely* 'no dysplasia' is a necessary condition for this radiologist to refrain from ordering a biopsy (or other significant short term follow-up). In other words, the physician's confidence in his/her interpretation depends on a mammogram free of significant and confounding background "noise." Otherwise, the radiologist would not risk the patient. Note that this strong and useful rule will not be generated by grid analysis systems that are not capable of producing appropriately hedged rules. The radiologist is given the confidence not to order aggressive follow-up partly by the fact that the mammogram is *extremely* free of dysplasia. The decision not to order a biopsy or other short term follow-up is one of the most critical and significant judgments in the interpretation of the mammogram; the consequences of error are serious. Novices are naturally and appropriately very cautious about such things, however, experts tend to call for significantly fewer biopsies, as a consequence, there is a considerable savings in time, money, and patient discomfort/worry. Consider the rule below:

(R2) IF *extremely* (1.0) no dysplasia
THEN *extremely* (1.0) confident in not ordering biopsy (.67).

This rule states that when the subject is extremely free of dysplasia, the radiologist is "usually" (.67) confident about not ordering a biopsy (or recommending short term follow-up), because significant lesions are not obscured by diffuse fibrocystic disease. However, the extreme lack of dysplasia also makes it easier to identify dangerous pathology (when it exists), and under those circumstances the radiologist can clearly see that a biopsy is required.

When these rules were presented to the domain experts, they expressed surprise at both the content of the rules and that the system could produce such rules. When confronted with rule (R1), the radiologists stated that it was true, but that they would not have been able to verbalize it (Chang, Ford, and Petry, 1987). This rule is an example of heuristic knowledge not found in books or taught in school, and may be an important part of what we characterize as expertise. Our preliminary experience suggests that the notion of physician domain experts developing medical expert systems directly (by abstraction of texts or interview) without the use of a knowledge acquisition tool and/or knowledge engineer (Tuhrim and Reggia, 1986) is overly optimistic. Rules such as those discussed above, which reflect true domain heuristic "expertise", would likely not be explicitly identified. What is actually needed therefore, is a knowledge acquisition tool which can efficiently and "naturally" interact with the domain expert; our experience suggests that the repertory grid approach is a promising example of such a tool with respect to the medical/radiology domain.

6. CONCLUSION

The major thrust of this research effort has been toward the development and unification of the necessary underlying theoretical foundations for an adequate approach to knowledge elicitation from repertory grid data.

A theory of non-quantitative confirmation is offered that incorporates the basic tenets of personal construct psychology directly into the logic as a grounds for the determination of relevance, thus strengthening the logic and extending personal construct psychology. This

Kellyan view of confirmation (i.e., relevance) may have broad application beyond the specific framework of this paper (see Ford, 1989). In fact, many scholars (Agnew and Brown, 1989; Fodor, 1983, 1987; Pylyshyn, 1987) maintain that the difficulty of establishing the relevance of available "facts" to a particular problem or hypothesis forms the crux of the *frame problem* – one of the most fundamental problems in AI. In any case, the development of an appropriate non-quantitative logic of confirmation was a necessary condition for the elaboration of an adequate probabilistic interpretation of quantitative confirmation.

The quantitative logic of confirmation stemming from the aforementioned work on relevance is also consistent with personal construct psychology in that it produces epistemic probabilities by the measurement of overlap of a person's constructs. Bundy's incidence calculus is extended and applied to the measurement of partial entailment thus affording a degree of rigor and formality. In this approach, the degree of confirmation is characterized by epistemic probabilities arrived at by measuring the overlap between the constructs' extensions represented as binary bit strings. This interpretation of the probability calculus permits a uniform, truth-functional method for measuring the degree of entailment associated with a proposition and provides a sound basis for the assignment of numeric values to expert system rules.

These largely theoretical developments are then applied to the representation and analysis of repertory grid data. The concept of an α–plane is introduced as a binary decomposition of grid data that furnishes the necessary realization of construct extensions (or ranges of convenience) needed to determine the range of relevance of a particular generalization or hypothesis. In addition, they provide the uniquely determined string of incidences required by any application of Bundy's truth functional incidence calculus. This incidence calculus enjoys numerous advantages in comparison with other mechanisms proposed for the automated handling of uncertainty in expert systems, but has previously suffered the drawback that although domain experts are sometimes able to provide a numerical certainty factor for a rule, they cannot assign the required string of incidences.

Finally, the theories elaborated in this research are applied to the design and construction of *NICOD* – a semi-automated medical knowledge acquisition system. In particular, the system has been successfully employed in the capture of valuable heuristic radiological knowledge (mammography) that the domain experts (radiologists) were otherwise unable to articulate.

7. REFERENCES

Adams-Webber, J. R. (1970). Actual Structure and Potential Chaos: Relational Aspects of Progressive Variations within a Personal Construct System. In: *Perspectives in Personal Construct Theory*, D. Bannister (Ed.), pp. 31-46. London: Academic Press.

Adams-Webber, J. R. (1979). *Personal Construct Theory: Concepts and Applications*. New York: Wiley.

Adams-Webber, J. R. (1984). Personal Construct Theory. In: *Encyclopedia of Psychology*, R. Corsini (Ed.), pp. 9-12. New York: Wiley Interscience.

Adams-Webber, J. R. (1987). Repertory Grid Technique: Research and Assessment. Invited lecture presented to Seventh International Congress on Personal Construct Psychology, Memphis, TN.

Adams-Webber, J. R. (1989). Some Fundamental Asymmetries in the Structure of Personal Constructs. In: *Advances in Personal Construct Psychology*, G. Neimeyer and R. A. Neimeyer (Eds.), Greenwich, CT: JAI Press Inc.

Agnew, N. M. and Brown, J. L. (1989). Foundations for a model of knowing: I. Constructing reality. *Journal of Canadian Psychology* (April).

Bainbridge, L. (1979). Verbal reports as evidence of the process operator's knowledge. *International Journal of Man-Machine Studies*, 11: 411-436.

Boose, J. H. (1984). Personal construct theory and the transfer of human expertise. *Proceedings AAAI-84*, pp. 27-33.

Boose, J. H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man-Machine Studies*, 23: 495-525.

Boose, J. H. (1986). *Expertise Transfer for Expert System Design*. Amsterdam: Elsevier Science Publishers.

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*, 26: 3-28.

Bundy, A. (1985). Incidence calculus: A mechanism for probabilistic reasoning. *Journal of Automated Reasoning*, 1: 263-283.

Chang, P. J., Ford, K. M., and Petry, F. E. (1987). PA 31: The production of expert system rules from repertory grid data based on a logic of confirmation. Talk presented to Seventh International Congress on Personal Construct Psychology, Memphis, TN.

Dreyfus, H. L. and Dreyfus, S. E. (1986). *Mind over Machine*. New York: Free Press.

Feigenbaum, E. A. (1977). The art of artificial intelligence 1: Themes and case studies of knowledge engineering. Report STAN-CS-77-621. Department of Computer Science, Stanford University.

Fodor, J. A. (1983). *Modularity of Mind: An Essay on Faculty Psychology*. Cambridge, MA: The MIT Press.

Fodor, J. A. (1987). Modules, frames, fridgeons, sleeping dogs, and the music of the spheres. In: *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*, Z. Pylyshyn (Ed.), Norwood, NJ: Ablex Publishing Corporation.

Ford, K. M. (1987). An approach to the automated acquisition of production rules from repertory grid data. Ph.D. diss., Tulane University.

Ford, K. M. (1989). A Constructivist View of the Frame Problem in Artificial Intelligence, An Invited Commentary. *Canadian Psychology* (April).

Ford, K. M. and Chang, P. J. (1989). An approach to automated knowledge acquisition founded on personal construct theory. In: M. Fishman (Ed.), *Advances in Artificial Intelligence Research*. Greenwich, CT: JAI Press.

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

Gaines, B. R. and Shaw, M. L. G. (1980). New directions in the analysis and interactive elicitation of personal construct systems. *International Journal of Man-Machine Studies*, 13: 81-116.

Gaines, B. R. and Shaw, M. L. G. (1981). New directions in the analysis and interactive elicitation of personal construct systems. In: *Recent Advances in Personal Construct Technology*, M. L. G. Shaw (Ed.), pp. 147-182. London: Academic Press.

Gaines, B. R. and Shaw, M. L. G. (1986). Induction of inference rules for expert systems. *Fuzzy Sets and Systems*, 18: 315-328. Amsterdam: Elsevier Science Publishers.

Haack, S. (1979). Do we need "fuzzy logic"? *International Journal of Man-Machine Studies*, 11: 437-445.

Hempel, C. G. (1965). Studies in the logic of confirmation. In: *Aspects of Scientific Explanation and Other Essays in the Philosophy of Science*, pp. 3-51. NY: Free Press.

Hersh, H. M. and Caramazza, A. (1976). A fuzzy set approach to modifiers and vagueness in natural language. *Journal of Experimental Psychology: General*, 105: 254-276.

Hetherington, P. A. (1989). Sureness measures and rating extremity in Repertory Grid Technique: An exploratory study. *International Journal of Personal Construct Psychology*, 1: 229-236.
Hoffman, R. R. (1987). The Problem of Extracting the Knowledge of Experts from the Perspective of Experimental Psychology. *The AI Magazine* (Summer).
Husain, M. (1983). To *What* Can One Apply a Construct? In: *Applications of Personal Construct Theory*, J. R. Adams-Webber and J. C. Mancuso (Eds.), pp. 11-28. New York: Academic Press.
Johnson, P. E. (1983). What kind of expert should a system be? *The Journal of Medicine and Philosophy*, 8: 77-97.
Kelly, G. A. (1955). *The Psychology of Personal Constructs*. New York: Norton Publishers.
Kelly, G. A. (1969). *Clinical Psychology and Personality: The selected papers of George Kelly*, B. A. Maher (Ed.), New York: Wiley.
Kelly, G. A. (1970). A brief introduction to personal construct theory. In: *Perspectives in Personal Construct Theory*, D. Bannister (Ed.), pp. 1-29. London: Academic Press.
Mamdani, E. H. and Efstathiou, J. (1984). An analysis of formal logics as inference mechanisms in expert systems. *International Journal of Man-Machine Studies*, 21: 213-227.
Neimeyer, R. A. (1985). *The Development of Personal Construct Psychology*. Lincoln: University of Nebraska Press.
Nicod, J. (1970). *Geometry and Induction*. Berkeley: University of California Press.
Osgood, E. O., Suci, G. J., and Tannenbaum, P. H. (1957). *The Measurement of Meaning*. Urbana, IL: The University of Illinois Press.
Plato. (1956). *Euthyphro, Apology, Crito*. Indianapolis, IN: The Bobbs-Merrill Company.
Pylyshyn, Z. W. (1987). Preface. In: *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*, Z. W. Pylyshyn (Ed.), Norwood, NJ: Ablex Publishing Corporation.
Quine, W. V. (1982). *Methods of Logic*. Cambridge, MA: Harvard University Press.
Salmon, W. C. (1973). Confirmation. *Scientific American*, 228: 75-83.
Schefe, P. (1981). On foundations of reasoning with uncertain facts and vague concepts. In: *Fuzzy Reasoning and its Applications*, pp. 189-215. London: Academic Press.
Sen, P. K. (1982). Approaches to the paradox of confirmation. In: *Logic, Induction and Ontology*. New Jersey: Humanities Press.
Shaw, M. L. G. (1980). *On Becoming a Personal Scientist*. London: Academic Press.
Shaw, M. L. G. (1982). PLANET: Some experience in creating an integrated system for repertory grid applications on a microcomputer. *International Journal of Man-Machine Studies*, 17: 345-360.
Shaw, M. L. G. and B. R. Gaines. (1987a). Techniques for knowledge acquisition and transfer. *International Journal of Man-Machine Studies*, 27: 251-280.
Shaw, M. L. G. and Gaines, B. R. (1987b). An interactive knowledge elicitation technique using personal construct technology. In: *Knowledge Acquisition for Expert Systems*, A. L. Kidd (Ed.), pp. 109-136. New York: Plenum Press.
Shaw, M. L. G. and Gaines, B. R. (1989). KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices. Accepted for publication in *International Journal of Man-Machine Studies*.
Shortliffe, E. H. (1976). *Computer-based Medical Consultations: MYCIN*. New York: Elsevier Science Publishers.
Slater, P. (1977). *Dimensions of Intrapersonal Space*, Vol. 2, P. Slater (Ed.), London: Wiley.
Swinburne, R. G. (1973). *An Introduction to Confirmation Theory*. London: Methuen & Co.
Tuhrim, S. and Reggia, J. (1986). Feasibility of Physician-developed Expert Systems. *Medical Decision Making*, 6: 23-26.
Von Wright, G. H. (1966). The Paradoxes of Confirmation. In: *Aspects of Inductive Logic*, J. Hintikka and P. Suppes (Eds.), pp. 208-218. Amsterdam: North-Holland Publishing Co.
Waterman, D. A. (1986). *A Guide to Expert Systems*. Reading, MA: Addison-Wesley.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

DESIGN REQUIREMENTS FOR KNOWLEDGE SUPPORT SYSTEMS

Brian R Gaines

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

ABSTRACT

The social role of information technology is analyzed in order to provide a framework for reasonable requirements for knowledge support systems. The many different scientific and engineering communities targeted on the development of specific knowledge support technologies are noted, and the problems of integrating developments from different communities are highlighted. The architectures of current integrated knowledge support systems are analyzed within the social framework and the strengths and weaknesses of current modules are evaluated. Major system developments in recent years are used to exemplify the analysis, and expected directions for future development are outlined. The framework, analyses and examples are used to define a structured requirements specification for future research and development related to knowledge support systems.

INTRODUCTION

This paper is the fourth in a series presented at previous AAAI Workshops on Knowledge Acquisition for Knowledge Based Systems which develop models of individuals and their role in the knowledge processes of society, and of the wide range of technologies supporting these knowledge processes (Gaines, 1987a, c, 1989a). Human 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. Computer-based systems, such as expert systems, knowledge acquisition systems and intelligent tutoring systems, play a supportive role in extending and enhancing these processes. This paper analyzes these knowledge support roles in detail, defining the functions of the technologies involved, describing the current state of the art, and developing a requirements specification for integrated knowledge support systems.

Figure 1 shows the way in which the specific negotiation process between an expert and a client is part of the general knowledge processes within the society of which they are part (Gaines, 1989a). The expert role is supported by, and accountable to, an appropriate professional community responsible for the management of the knowledge processes within the relevant profession. The client role is supported by, and accountable to, an appropriate client community responsible for the identification of problems and the provision of resources within the relevant applications domains. The concept of a 'role' is emphasized because individuals play both expert and client roles within many different communities. Neither 'roles' nor 'communities' are absolute distinctions—they each represent useful simplifications of the, otherwise unmanageable, complexity of social processes.

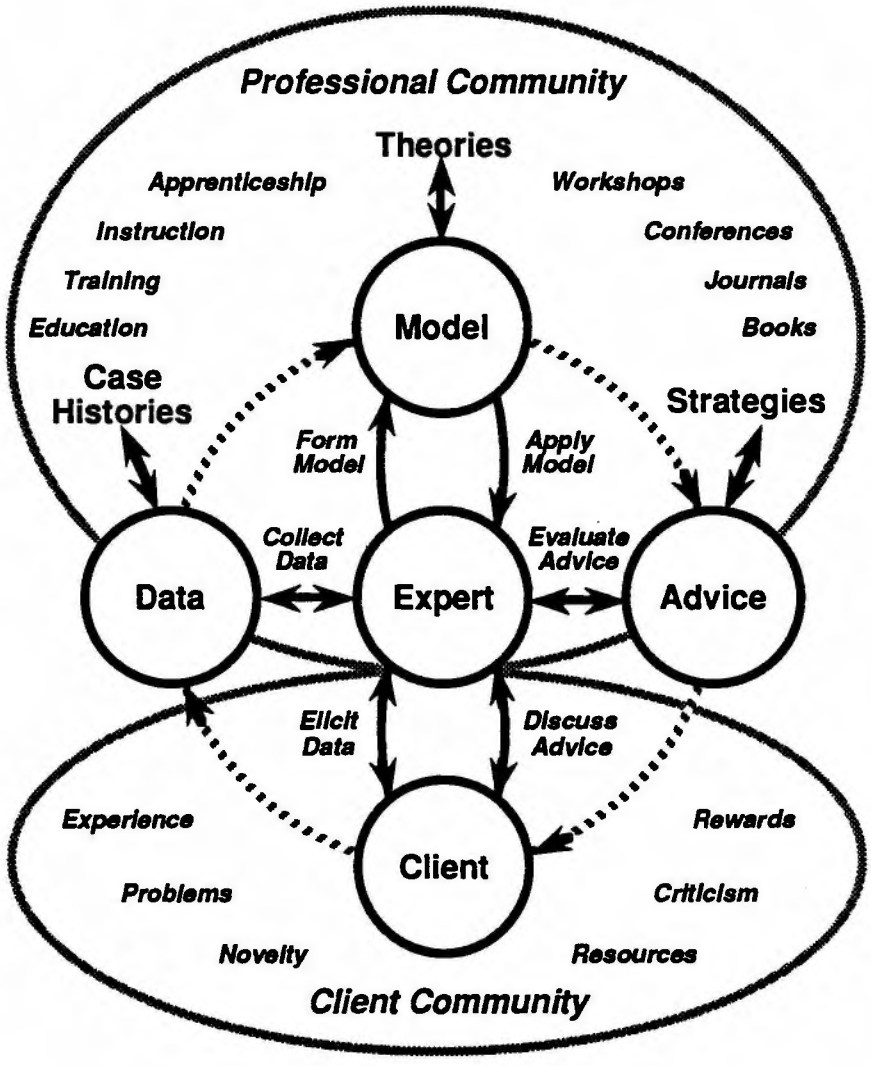


Fig.1 The social context of knowledge support processes

The negotiation cycle at the center of Figure 1 captures Hawkin's (1983) model of client-expert interaction. The expert elicits data about the problem from the client, forms a model based on this data, generates advice based on this model and feeds this back to the client. The client may query the advice and, possibly, the model, and the reasons for such a query lead to further data elicitation, and repeat of the elicitation/modeling/advice/query cycle. In this process the expert may collect new cases, form new models, and generate new advice-giving strategies, thus increasing his or her expertise.

The social mechanisms in the client community constitute positive feedback loops encouraging the concentration of individual expertise by giving established experts greater access to novel problems (Gaines, 1988c). The social mechanisms in the professional community constitute negative feedback loops encouraging the dissemination of individual expertise throughout that community.

This paper extends the model of Figure 1 to incorporate existing knowledge-based system technologies with a view to analyzing their role in social knowledge processes, and their strengths and weaknesses in fulfilling it. This analysis is used to develop a requirements specification for knowledge support system development during the next decade.

KNOWLEDGE SUPPORT SYSTEMS

In analyzing the current state of the art in knowledge-based system technologies, it is interesting to look back at a previous generation of information technologies supporting human decision processes and see how our requirements have progressed. Keen and Morton (1978) define a *decision support system* as one using a computer to:

- Assist managers in their decision processes in semistructured tasks
- Support, rather than replace, managerial judgements
- Improve the effectiveness of decision making rather than its efficiency.

These look remarkably like the expectations for 'expert systems' since their inception. The only significant extension is that we expect an expert system to have some kind of 'explanation' capability, to:

- Account for the reasoning underlying advice given.

If we adopt a general definition of a 'manager' as a person responsible for activities in areas where we do not have effective, overt models, then the requirements above capture the general stereotype of an 'expert system.'

However, our expectations of knowledge-based systems in general have progressed well beyond this minimal list in recent years:

- The *knowledge acquisition* community has generated requirements for systems to incorporate knowledge elicitation and analysis modules automating expertise transfer (Boose & Gaines, 1988; Gaines & Boose 1988).
- The *machine learning* community has generated requirements for systems to incorporate inductive modeling modules automating empirical induction (Michalski, Carbonell & Mitchell, 1986).
- The *case-based reasoning* community has generated requirements for systems to incorporate comparison modules automating analogical reasoning (Kolodner, 1988).
- The *natural language* and *machine translation* communities have generated requirements for systems to incorporate linguistic analysis and generation modules automating language understanding (Cullingford, 1986; Lehrberger & Bourbeau, 1988).
- The *knowledge representation, explanation-based learning* and *symbolic reasoning* communities have generated requirements for systems to incorporate deep reasoning modules automating reductionist derivation (Cercone & McCalla, 1987; Hobbs & Moore, 1985).
- The *intelligent tutoring* community has generated requirements for systems to incorporate user modeling modules automating individualized knowledge transfer (Wenger, 1987).
- The *electronic mail* and *teleconferencing* communities have generated requirements for systems to incorporate communication modules automating inter-person communication (Vervest, 1987; Hiltz, 1984).
- The *electronic publishing* community has generated requirements for systems to incorporate typographic and graphic modules automating knowledge presentation (Vliet 1988).
- The *information retrieval* community has generated requirements for systems to incorporate knowledge indexing modules automating contextual retrieval (Oddy, Robertson, van Rijsbergen & Williams, 1981).
- The *visual programming* community has generated requirements for systems to incorporate interactive graphic modules automating visual interaction (Shu, 1988).
- The *hypermedia* community has generated requirements for systems to incorporate knowledge linkage modules automating associative thinking (Barrett, 1989).
- The *computer support of cooperative work* community has generated requirements for systems to incorporate knowledge-sharing modules automating group collaboration (Greif, 1988).



Fig.2 Computer-based knowledge support processes

Figure 2 shows some of the relations between a knowledge structure and other knowledge structures in the world, the mind, media and computers. The central knowledge structure might represent any of those shown peripherally so that the diagram captures a wide variety of knowledge transfers and transformations. The lower part of the diagram is largely concerned with knowledge application and the upper part with knowledge acquisition. Additional activities have been added such as the computer support of creativity which was the objective of Smith's (1977) *Pygmalion* system and of the Japanese *Sixth Generation* research proposal (STA, 1985; Gaines, 1986a), but does not seem to be a major current research focus. Moving around the lower part of the diagram from left to right there are a variety of user roles that correspond to various processes in the professional and client communities of Figure 1. To a large extent the ones to the right build on, and subsume, those to the left—for example, all roles shown are part of the essential serendipity (Roberts, 1989) of the creative process. Even if the system is not designed for the activities to the right it is unavoidable that it be used to support those roles since users will be involved in them regardless of the system designers' intentions. However, the effective support of any of the roles shown requires careful system design based on an understanding of the knowledge processes involved.

The figure and the list above are by no means exhaustive. There are communities concerned with numeric data processing, data bases, simulation, instrumentation and control, person-computer interaction and so on, which are also components of knowledge-based systems. The scientific *communities* involved in specifying these requirements are emphasized because this draws attention to the current fragmentation of research on knowledge-based systems. There have been a few major projects that cross these communities and combine requirements, notably the MYCIN/TEIRESIAS/ GUIDON developments (Buchanan & Shortliffe, 1984; Clancey & Shortliffe, 1984), but the majority of developments fall within particular communities and focus on narrowly defined objectives. This division of labor has been a major strength in developing specific knowledge technologies, but a counter-balancing integration of the requirements and techniques for achieving them is becoming increasingly important. Each area has far to go in its specific developments, but each of them is also facing limitations through problems where potential solutions are in other areas.

The term *module* is used deliberately because it is symptomatic of problems in developing integrated systems that there are now modules without 'modularity.' Current modules are generally designed as stand-alone systems where open-system considerations (Slonin, Schonbach, Bauer, MacRae & Thomas, 1987) of inter-operability, portability and scalability are not significant requirements. These considerations are fundamental meta-requirements if advances in knowledge-based system technologies in each of the specialist communities are to migrate across boundaries, supporting developments within each community, and also supporting the development of integrated, large-scale, knowledge-based systems. *Integratability* is itself a significant technology with its own methodologies and techniques. A new world of system integration possibilities would be opened if every system developed was split into functional modules which could be called as libraries by other modules.

The generic term *knowledge support system* is used to encompass all systems whose primary function is to support knowledge processes in society. The primary focus in this paper is on computer-based knowledge support systems, but the term is deliberately left unqualified to encompass non-computer-based systems supporting knowledge processes. Effective relationships with existing media and social phenomena supporting knowledge processes form another major meta-requirement for computer-based knowledge support systems. Information technology is a by-product of what Beninger (1986) terms the 'control revolution' taking place from the mid nineteenth century to today. He views this as a necessary social response to the complexities of global economic processes whose problems of instability were identified by Durkheim (1893). This revolution and its supporting technologies are themselves intermediate products of the longer-term evolution of human civilization and its supporting knowledge processes (Gaines & Shaw, 1986; Wojciechowski, 1986).

To view information-technology as a by-product of social processes is not to deny its role as an enabling technology generating change in those processes. The notions of cause and effect are only convenient perspectives from which to view the complexities of the total social system. Innis (1972) has analyzed the profound role of media in generating social change throughout recorded civilization, and Meyrowitz (1985) has given a deep analysis of the impact of electronic media on social behavior. Heim (1987) has given a thought-provoking study of the impact of the word processor on academic scholarship and the underlying conceptual structures. Knowledge support technologies are both a response to social needs and an agent of social change. As Popper (1968) has noted knowledge is produced by people for particular purposes, but once it exists it has unintended and unexpected consequences making it quasi-autonomous.

It is interesting to read the recent literatures of the various communities listed above and discern the commonality of their underlying objectives. For all the achievements of modern information technology, it has yet to deliver the level of support of human knowledge processes that have excited those involved in its development. This is a healthy source of frustration and continuing challenge, but the frustration will not be overcome and the challenge met by developments in one community alone. It is the combination of the current and coming developments from each community in integrated knowledge support systems that is necessary to satisfy the aspirations in each community. Sooner or later an effective 'divide and conquer' strategy has to be transformed into an 'integrate and evolve' strategy. Perhaps we and the technologies are not yet ready, but unless we continue to test that readiness the frustration will continue and the larger challenges will not be met.

The next section puts some critical technical developments within the social context of Figure 1 in order to analyze the current state of the art in knowledge support systems and identify reasonable requirements for the next stage of research.

KNOWLEDGE SUPPORT SYSTEMS

Figure 3 shows the architecture of a knowledge support system standing between the professional and client communities of Figure 1 as an intermediary between experts and clients. The detailed architecture captures the current structure of knowledge acquisition tools, knowledge bases and expert system shells, extended to allow for the integration of hypermedia and intelligent tutoring concepts. The shaded modules indicate areas where our understanding, methodologies and tools are currently relatively weak. The role of the 'knowledge engineer' shown on the right links back to earlier arguments that this role is no longer one of an *intermediary* between expert and knowledge-base, but rather one of *managing* the knowledge acquisition and overall system development processes (Gaines, 1987b). The knowledge engineer and the technologies for knowledge acquisition, advice giving, explanation, and tutoring, are part of the *knowledge technologies community*.

The top oval encompassing the *knowledge acquisition system* in Figure 3 contains tools for eliciting knowledge from experts and for analyzing the knowledge to create a knowledge base. The intermediate knowledge bases elicited include the experts' conceptual structures and problem-solving strategies, and sample cases in the domain. The analysis tools are also used to feed back this knowledge to the experts in various forms for checks of coherence and consistency, and to prompt the experts to explore and structure their knowledge. Shaw and Woodward (1989) have analyzed and tabulated existing knowledge acquisition methodologies and techniques in terms of their elicitation and analysis components, and their relative emphasis on the elicitation of problem structure versus problem-solving strategy. Boose (1989) has noted the weaknesses of current knowledge acquisition tools in eliciting problem-solving strategies which limits their application in synthesis tasks such as design and planning.

The bottom oval encompassing the *expert system shell* in Figure 3 contains tools for making inferences from general knowledge about specific cases from clients and for interacting effectively with the clients. The intermediate interactions generated include queries to define the specific case, advice about appropriate action and explanations about the basis for such advice. The analysis tools are also used to feed back this knowledge to the experts in various forms for checks of coherence and consistency, and to prompt the experts to explore and structure their knowledge. Clancey (1987a) has noted the weaknesses of explanation facilities in current expert system shells which limits their application in knowledge transmission tasks such as tutoring.

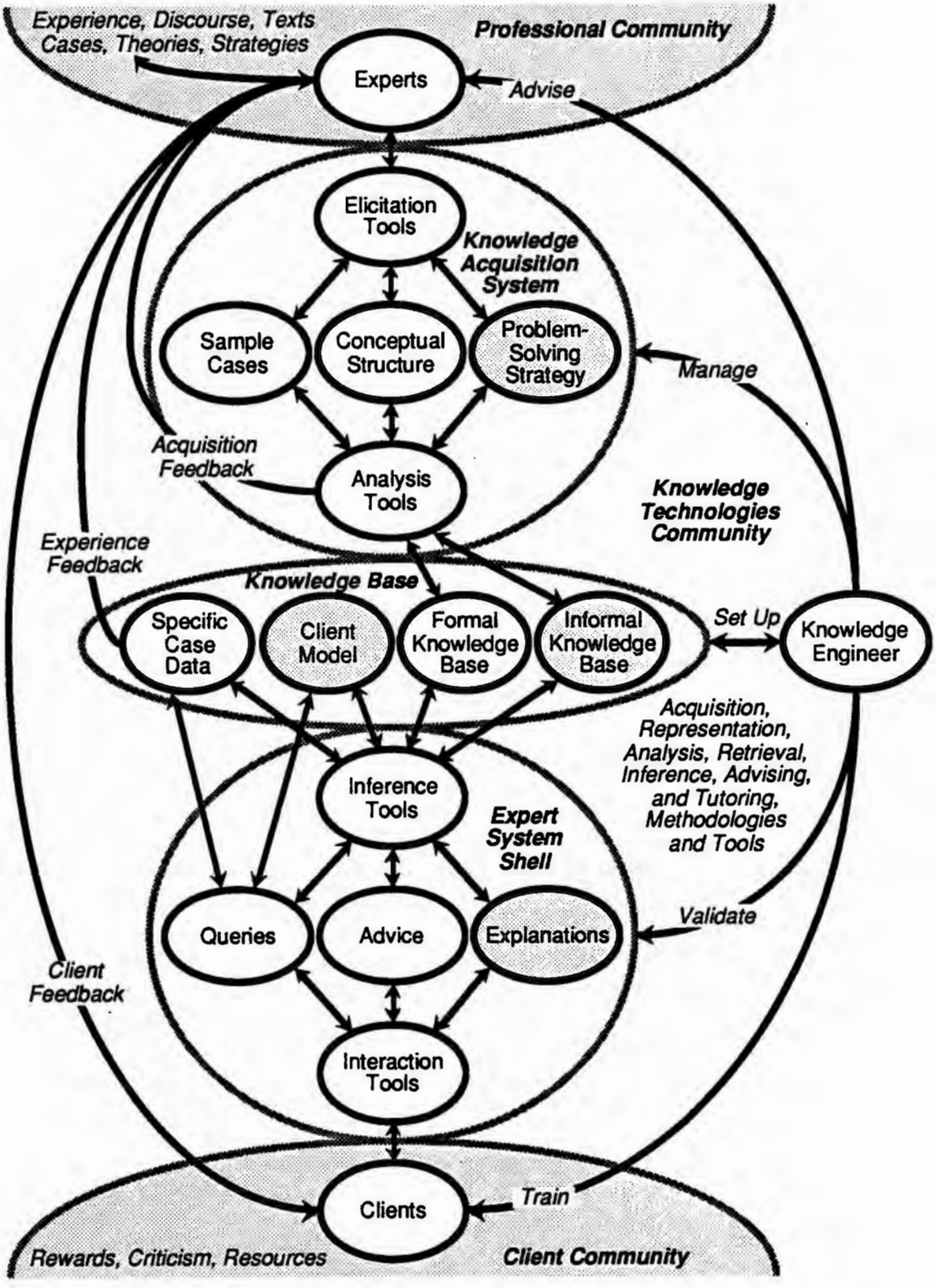


Fig.3 Knowledge technologies within the social context of client-expert relations

The central oval encompassing the *knowledge base* in Figure 3 contains four components: a *formal knowledge base* capturing knowledge structures that may be used computationally as part of a formal inference process; an *informal knowledge base* capturing knowledge that may be used for annotation and explanation but not for computation; *specific case data* that may be used to model the client's situation in a form suitable for advice generation; and a *client model* that may be used to target advice and explanations to the needs and capabilities of particular clients.

The formal knowledge and specific case data components are part of all expert system shells. The informal knowledge base is a common extension of current shells through links with hypermedia tools in order to support knowledge processes such as association and visualization that do not depend on formal inference but support human understanding of such inference. The client model is a research extension of current shells to provide user modeling supporting intelligent tutoring systems and intelligent user interfaces. It is reasonable to expect that most interactive systems supporting knowledge-based discourse with clients will need to have some form of client model not only to cope with diversity in the community of clients but also to cope with the varying knowledge states of individual clients.

It is noteworthy that the flows of information in the situation portrayed in Figure 3 are such that the specific case data and client model both necessarily depend on client interaction with the system, either through queries originating from the inference tools in the shell or from dialog initiated by the client. The system only knows about the case as described by the client. The accuracy of its inference procedures and resultant advice are critically dependent on the client understanding the conceptual structures of the expert sufficiently to answer queries correctly (Shaw & Gaines 1988). Similarly, the system only knows about the client as modeled through discourse with the system and hence the richness and accuracy of the client model are critically dependent on what may be a very limited communication.

The relations between the professional and client communities after the introduction of knowledge-based system technologies can also be analyzed in terms of the feedback paths on the left of Figure 3. At the upper left, *acquisition feedback* from the knowledge acquisition system to the expert during the development of the formal and informal components of the knowledge base is well-established practice. At the center left, *experience feedback* from the application is shown as an equally important but less well-supported feature. If the expert, after the acquisition phase, is completely detached from the clients' experiences in using the knowledge-based system it can seriously undermine his or her role as a manager of the inductive process, and the related development of the professional community. A significant requirement for any knowledge support system is that it capture case data, particularly anomalous cases, and feed this back to the expert (Gaines, 1988c) as shown in the experience feedback path.

At the lower left, even if the experience feedback path is totally effective, direct *client feedback* through expert-client interaction is still necessary if the case falls outside the conceptual structure that has been elicited from the expert. It may be necessary to transcend the discourse between client and expert supported by the system and initiate another level of communication that may result in major changes to the overall system. In any event, in a professional community the need to support and interact with clients outside the constraints of formal relationships imposed by any means, including technology, is well-recognized as crucial to professional development (Schön, 1983).

The role of the expert within the professional community as a recipient of new knowledge disseminated within that community, as shown at the top of Figure 3, emphasizes the need for knowledge acquisition to be an ongoing process, not a one-off development.

As emphasized previously, the distinctions between experts, knowledge engineers and clients in Figure 3 is one of roles, not individuals (Gaines, 1987c). In particular, the expert and client roles may be played by the same person at different times. It is clearly good practice for the expert to act as a client for the system incorporating his or her expertise. If the knowledge acquisition system is embedded in the advisory system, it is feasible for the knowledgeable client to assume the role of expert and post an anomalous case edited to have the correct advice to the database of sample cases and re-run the analysis tools to generate a new knowledge base (Gaines, Rappaport & Shaw, 1989). The distinctions made in Figure 3 do not preclude such activities but they are significant none the less because the *responsibilities* and *accountabilities* are different for the same individual in his or her different roles.

THE STATE OF THE ART

There are now many systems that implement part of Figure 3 but few that attempt to encompass the complete structure shown. It is interesting to analyze the more complete systems in terms of their structure, achievements, strengths and weaknesses, with a view to determining the current state of the art.

The outstanding example of a relatively complete implementation of a knowledge support system with the architecture of Figure 3 is the MYCIN/TEIRESIAS/GUIDON system already referenced. This has also been extremely well-documented by its developers (Shortliffe, E.H., 1976; Davis, 1982; Buchanan & Shortliffe, 1984; Clancey & Shortliffe, 1984; Clancey, 1987a, b, c) and others (Wenger, 1987; Evans & Patel, 1989) so that the analysis can be extended as far as required.

In terms of the lower oval in Figure 3, the EMYCIN variant of MYCIN is the original expert system shell on which later developments have been modeled. Its explanation facilities are based on the presentation of the rules used in deriving its conclusions, and this has again been a model for later systems. The main developments since have been in more structured knowledge representations for expressing the experts' conceptual structures such as frame and object representations which allow more concise and expressive inference rules to be used. Research developments include different forms of inference, particularly case-based reasoning based on direct comparisons with the sample cases rather than than dependence solely on inference rules. None of these changes affects the potential system advisory performance since structured representations can be flattened and case comparisons can be expressed as rules, but they do significantly affect the discourse possible with the system by both expert and client.

In terms of the upper oval in Figure 3, the TEIRESIAS component of MYCIN provided the primary knowledge acquisition system. It has some remarkable features which do not appear to have been replicated in later systems, notably an ability to guide debugging of the knowledge base on anomalous cases through meta-rules, and primitive capabilities to translate restricted natural language to rules for acquisition and rules to natural language for acquisition feedback (Davis 1982). The meta-rules also provide a vehicle for the capture of problem-solving strategies, but this was applied primarily to debugging strategies rather than diagnostic strategies.

MYCIN and TEIRESIAS supported the advisory and developmental activities of the system viewed as a knowledge support system, and it was natural to suppose that this support role could be extended to tutoring. However, in developing GUIDON as an intelligent tutoring system based on MYCIN, Clancey (1987a, b, c) found it necessary to radically restructure both the knowledge base and inference tools resulting in NEOMYCIN, and to add a modeling capability to keep track of the students' knowledge states as they played out their own strategies in attempting the diagnostic task.

The knowledge base restructuring was to some extent a decompilation from decision rules back to the knowledge underlying them, and to some extent the addition of knowledge irrelevant to performance but significant in explanation. The student modeling capability involved a separation of problem-solving strategies, particularly hypothesis management from the domain knowledge. The direction of development of expert system shells towards object-oriented databases has made it easier to represent knowledge through declarative structures leading to decisions through the application of strategic meta-rules. The choice of cognitively natural structures and strategies is still a major problem. Archetypal research focusing on the ontology of knowledge is the KADS project in Europe (Breuker & Wielinga, 1987), and archetypal research focusing on problem-solving strategies is the 'generic task' approach (Chandrasekaran, 1988) capturing the methodologies underlying many of the problem-solving methods used in significant expert system developments (McDermott, 1988).

It is easy to assume that advances in technology supporting an improved system that are the end point of one project are the natural starting point for further projects. However, Clancey (1987a), in summing up the GUIDON development experience, sounds a note of caution whose rationale is very significant to future requirements:

> "Despite the now apparent shortcomings of MYCIN's rule formalism, we must remember that the program was influential because it worked well. The uniformity of representation, so much the cause of the 'missing knowledge' and 'disguised reasoning' described here, was an important asset. With knowledge so easy to encode, it was perhaps the simple parametrization of the problem that made MYCIN a success. The problem could be built and tested quickly at a time when little was known about the structure and methods of human reasoning. ... We can treat the knowledge base as a reservoir of expertise, something never before captured in quite this way, and use it to suggest better representations."

These remarks can be read not only as a commentary on the state of the art in knowledge-based systems but also, contrapuntally, as an insight into cognitive processes whereby knowledge is expressed and iteratively refined. The elegance and explicability of more structured knowledge representations do not imply that they are necessarily more appropriate for knowledge acquisition than other representations that may be closer to the 'compiled' knowledge structures underlying skills. In terms of requirements, it is not a question of right and wrong representations, but rather a reasonable plurality of representations with usable tools to move between them.

The significance of the MYCIN-NEOMYCIN transition for knowledge support system architectures is not so much that intelligent tutoring should be supported, but that effective explanation may involve knowledge structures over and above those required for effective performance. There is a range of behaviors we may expect from a knowledge support system that involve different operational criteria and performance evaluation. Information retrieval requires effective indexing but no computation with the stored material. Advice giving requires effective inferencing but no communication of the basis of the advice. Explanation, as an auditing activity, involves effective communication of a trace but no justification of the steps in the trace. Explanation, as a tutoring activity, involves justification of reasoning within a framework of a student's existing knowledge but no overt attempt to promote creative thinking.

These, and other, operational modes are reasonable requirements for knowledge support systems, but the concept of knowledge 'acquisition' begins to break down if we consider it as primarily as a transfer process leading to a single underlying knowledge base. A system adequate to operate in all these modes will involve multiple levels and representations of knowledge and will be in a state of continuous refinement.

Other system developments targeted on a knowledge support role have shown similar patterns of development to that analyzed above. The transition from ETS (Boose, 1986) to AQUINAS (Boose & Bradshaw, 1987) has been one of increasing the richness of knowledge structures, including extensions to manage and utilize multiple sources of expertise, to enhance the case-based reasoning tools, and to add an intelligent dialog manager automating some activities of the knowledge engineer (Kitto & Boose, 1987). The heterogeneous integration of KSS0 (Gaines & Shaw, 1987) as a knowledge acquisition tool, with NEXPERT (Rappaport 1987) as an expert system shell supporting the formal knowledge base, and HyperCard (Goodman, 1987) as a hypermedia system supporting the informal knowledge base, has created an integrated knowledge support system in which acquisition and performance are complementary continuing activities (Gaines, Rappaport and Shaw, 1989). Musen's (1989) systems for knowledge acquisition for clinical trials exemplify the power of a multi-level, meta-systems approach.

The major problem in all these extensions and enhancements has been to maintain an essential simplicity in application—to increase functionality without loss of usability. This requires a system architecture which is initially simple but unfolds to offer increasingly complex capabilities in response to the growing needs and skills of the user. It might be this is best achieved through a series of distinct tools, for psychological as well as technical reasons, but it is also a challenge to achieve this responsiveness within a single, coherent architecture.

THE ROLE OF HYPERMEDIA

The one module in Figure 3 that the GUIDON developments did not address is the 'informal knowledge base' on the right. Presumably, the reasons for the this were that the focus of attention was on formal knowledge structures and also that hypermedia technology was not in a sufficiently advanced state. It is an interesting thought experiment to imagine a MYCIN development within a 'library and information science' ethos in which medical text books and journal articles are seen as the paramount vehicles for knowledge. Effective information presentation, including decision trees, and hypertext linkage to relevant background information, clinical trial data, counter-indications, and so on could be used to provide a system supporting medical scholarship for those at various levels of expertise. The IRIS Intermedia applications experience suggests that such an approach does effectively support the knowledge processes involved, including knowledge transfer (Yankelovich, Landow & Heywood, 1987).

What would be missing in such a system would be the capability of the system to act as a concise advisor, asking appropriate questions and making diagnoses. However, adding OPS5 or Prolog rules to the informal knowledge base in such a way that they 'shadow' corresponding textual statements, and are mutually accessible through hypertext links, creates a mechanism for conventional 'expert system' consultations. Figure 4 shows the shadowing of the informal by the formal as a system architecture. In fact, systems with such facilities already exist but they have been created through the reverse process, of annotating the formal structures of a knowledge-based systems with hypertext links.

The significance of this thought experiment is not in the technology, but in where it places the definitive knowledge structures. Books, journals, diagrams and pictures are still our prime repositories of knowledge. Formal knowledge structures supplement them by allowing them, in part, to become operational which is a significant extension. However, operationalism is a supplement to communication, not a replacement for it. A person who does something for you when you ask them to show *you* how to do it has a very limited role repertoire. Human society is robust in its functioning because it has developed a very wide range of forms of interaction and levels of discourse.

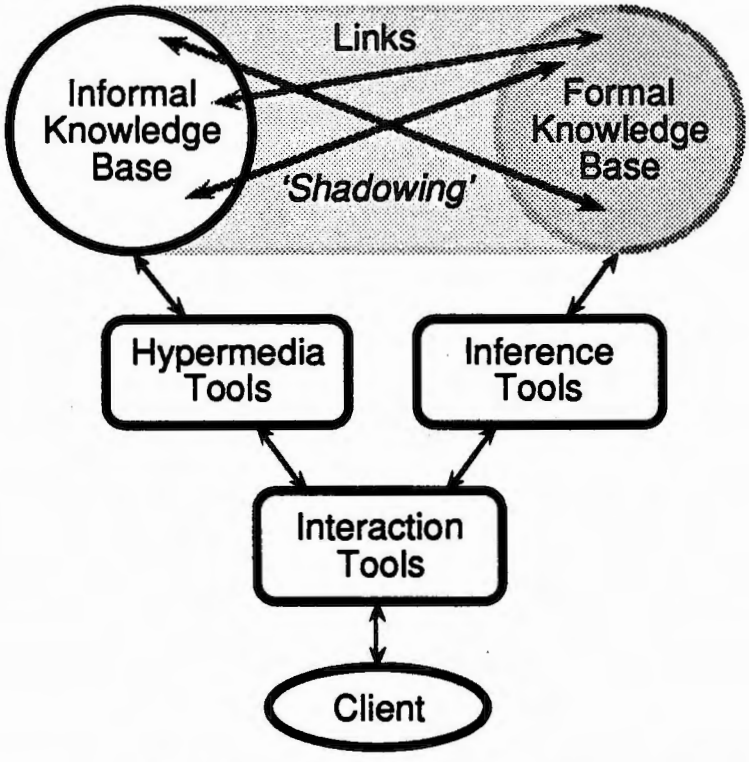


Fig.4 Shadowing informal knowledge with formal knowledge

Sergot (1988) has addressed the relationship between informal and formal knowledge structures when assessing the various legal expert systems that have been developed. He points out that the apparent clarity of law when expressed in logical rules is misleading. It is a key rule in the interpretation of legal documents that the exact wording is critical so that paraphrasing is forbidden. Yet the computational structures of the expert system are inherently a paraphrase of the legal text in a formal language. Sergot also notes some paraphrases involve more problems than others. Much legal expertise relates to the interpretation of written law, and it is good practice within the formal knowledge base to keep the more volatile interpretative rules separate from the more stable formalization of the written legislation.

These considerations are ubiquitous, not a peculiarity of the legal system. The facts of scientific experiments, such as clinical trials, and our interpretations of them are distinct knowledge structures. Both are theory-laden, in that our conceptual frameworks determine what we can express. The written descriptions, instrumentation records, photographs and diagrams with which we describe both are the paramount repositories of knowledge. Formal, operational translations of these may also attain a similar status but the formal and informal knowledge structures do not determine one another, they act only as mutual constraints. In particular, interpretations are volatile and subject to reconceptualization.

This is the essence of scholarship, explanation and tutoring, and also of the role of the expert as a manager of the reflective equilibria between cases and explanations, facts and theories, decisions and rationalizations. The formal and informal knowledge bases in Figures 3 and 4 are not alternatives or separated through technological distinctions. They are both essential and are linked through subtle relationships which are derived from the essential nature of knowledge. The types of link that will exist in a knowledge base are shown in Figure 5.

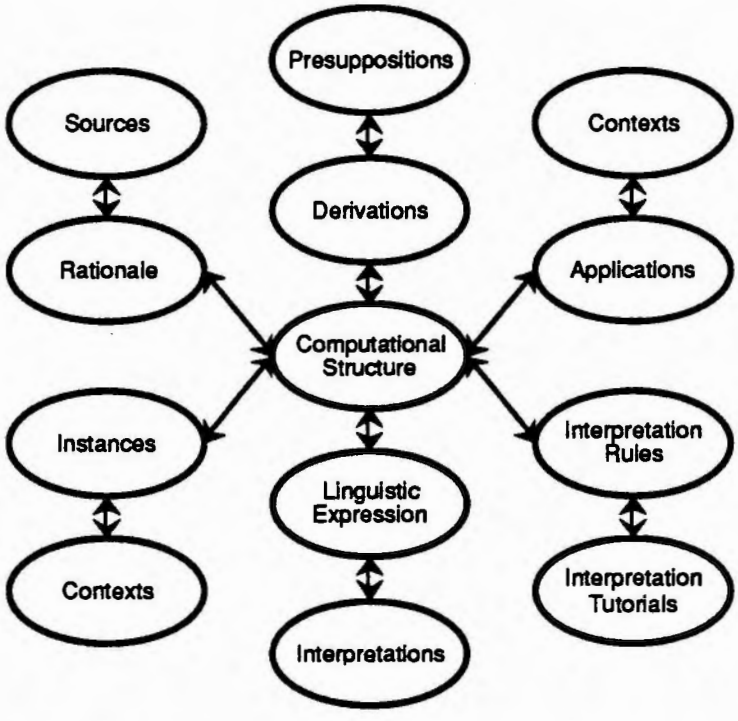


Fig.5 Links between informal and formal knowledge

In the center of Figure 5 is some computational structure which may be as small as a single rule or as large as a complete expert system—annotation is necessary at all levels and varies in nature at different levels. Immediately below is the linguistic expression of the structure, possibly quoted from some document which is itself the definitive statement of the knowledge. This in turn is linked below to interpretive commentaries. Above the computational are various alternative derivations of it which themselves depend on presuppositions to which they are linked. At the top left are various rationales for the structure linked back to their sources—these rationales may involve references to the other structures such as derivations. At the bottom left are instances of the knowledge structure linked into contexts in which they can occur. At the top right are applications of the knowledge structure again linked to their contexts. At the bottom right are the explicit interpretation rules for the terms within the computational structure linked to tutorial material to support the client in using these terms consistently with the system.

Figure 5 presents only an example of significant possibilities and omits the many cross-reference links that will occur. The use of labeled links to carry knowledge structures was an important part of Trigg's (1983) original Textnet design, and Van Lehn (1985) has described a case history in which the development of such links led to changes in knowledge structures in a discipline. Hypermedia developers have emphasized the support of knowledge processes through the user chasing the informal links, whereas expert system developers have emphasized the computer chasing the formal links. The two approaches are complementary and can be coupled within their existing technical frameworks and existing user interfaces (Gaines & Vickers 1988).

The short-term problems are technical ones of having the appropriate 'hooks' available in expert system and hypermedia packages to support communication between them so that they appear as a seamless single application to the user. Often this can be achieved when it is not part of the original design, but more powerful and elegant implementations can be achieved if an open systems approach is adopted from the start.

REQUIREMENTS FOR KNOWLEDGE SUPPORT SYSTEMS

Having highlighted developments stemming from the MYCIN project as illuminating key aspects of knowledge support system requirements, it is appropriate to commence drawing up a more explicit specification by listing the requirements specified by MYCIN users, that the system should (Clancey & Shortliffe, 1984 p.464):

- Be able to explain their decisions to users
- Be portable and flexible so that users can access them at any time and place
- Display an understanding of its own knowledge
- Improve cost-efficiency
- Automatically learn new information when interacting with experts
- Display common sense

and by the system's developers, that research should focus on (Clancey & Shortliffe, 1984 p.467):

- Psychological studies providing new insights into simulating expert decision-making
- Techniques for representing and using causal and mechanistic relationships allowing reasoning from first principles
- Methods for acquiring expert knowledge, encoding it and checking it for consistency and completeness
- Explanation facilities guided by understanding of how people explain things to one another and adapt to the knowledge and experience of the person requesting advice
- New machine architectures supporting high-performance decision-making programs
- Melding of symbolic techniques drawn from artificial intelligence and analytic techniques of statistics, pattern recognition and decision theory
- Novel ways in which personal computing and graphics might improve the acceptability and cost-effectiveness of systems aiding decision-making tasks.

These requirements include the 'second-generation' (Gaines 1988a) knowledge-based systems concepts that systems should be able to learn through deep reasoning (Steels 1986) and empirical induction, an emphasis on knowledge adequate to support explanation and understanding, a need to understand processes of human expertise and knowledge acquisition, and considerations of new technologies enabling system improvement in a variety of ways. They omit considerations of the embedding of expert systems in society and the embedding of expert systems in larger-scale information systems.

Is it possible to develop a 'requirements specification' for knowledge support systems that captures the diversity of issues addressed in this paper and elsewhere, and yet does not become so all-embracing as to be meaningless? I believe that it is both reasonable and necessary to begin to attempt to do so. Reasonable, because there are now enough development and application examples to ground the specification in significant experience. Necessary, because large-scale system developments require clear objectives and careful management if they are result in effective sustainable applications.

One of the problems we have to face is that any requirements specification for knowledge support systems is bound to apply to all information technology, that is, we can not differentiate something that is specifically a 'knowledge-based' system. Knowledge support systems carry with them all the problems and requirements of conventional information technologies plus additional requirements that stem from greater expectations about their capabilities and roles. However, success at the conventional level, particularly in areas of open-architecture, user interfaces and interactivity, is essential to success at the other levels peculiar to knowledge support systems. Some general technical considerations are too important not to be mentioned, although they are non-specific.

THE TOP-LEVEL GENERATIVE META-REQUIREMENTS

The overall consideration from which all requirements for knowledge support systems can be generated is that they be specified in terms of their roles in the social knowledge processes shown in Figure 1. This is in contrast to specifying them in technical terms or individual cognitive terms as is common currently.

Overall Requirement: A knowledge-based system is a component in a structure of social knowledge processes. Specify the top-level requirements in terms of the relevant structure, existing knowledge processes, intended changes and unexpected side-effects.

Some meta-requirements may be derived immediately from this in terms of the generic social mechanisms shown in Figure 1.

Meta-Requirement 1: The social structure usually involves a professional community responsible for managing the processes of knowledge acquisition and dissemination and a client community dependent on the knowledge for its activities. Each community has existing requirements for knowledge processes that need to be satisfied by any knowledge support system. Specify these as part of the system requirements.

Meta-Requirement 1.1: The professional community has procedures for managing knowledge acquisition and dissemination. Specify that these should be supported, not by-passed, by the knowledge support system. For example, the 'experience feedback' and client 'feedback' paths in Figure 3 should be explicitly recognized and supported.

Meta-Requirement 1.2: The professional community has procedures for assessing the risk of using new knowledge processes, controlling resulting hazards, and ensuring proper accountability. Specify that these should be supported, not by-passed, by the knowledge support system.

Other meta-requirements may be derived by considering the role of knowledge in terms of the generic social mechanisms shown in Figure 1. Knowledge is valuable, preserved in many forms, fallible, constantly changing, and created through a wide variety of processes, some intrinsically irrational.

Meta-Requirement 2: A knowledge support system will not contain all the knowledge relating to the processes in which it is involved, or provide all the facilities required. Specify how the system will relate to existing knowledge support systems including non-computer media and information technology.

Meta-Requirement 2.1: Specify the system to have an open architecture such that other systems are readily embedded in it, and it is readily embeddable in other systems.

Meta-Requirement 2.2: Specify field enhancement, upgradability, scalability and portability of functionality as essential requirements. Specifying the automation of these leads to various forms of embedded learning.

Another class of meta-requirements that are peculiar to knowledge-based systems is to note the intrinsic reflexivity of knowledge processes, that they apply to themselves, and require that the system support this.

Meta-Requirement 3: Knowledge processes are intrinsically reflexive, applying to themselves. Require that this reflexivity be explicitly supported by models within the system of itself, its users, its context, its processes and its tasks.

Meta-Requirement 3.1: Specify that the system should be able to explain its own operation.

Meta-Requirement 3.2: Specify that the system should be able to adapt to the differing knowledge states of its users.

Further refinement and instantiation of these meta-requirements is in itself adequate to generate specifications for the various sub-processes shown in Figure 2.

Figure 6 shows a system architecture for a large-scale knowledge-support system in an industrial context that has the capability to satisfy these requirements through integration of existing heterogeneous systems (Gaines 1988b). The lowest level is a network layer supporting open system integration. The middle level is a combination of user interface and information systems with virtual circuits between them, that is, current information technology. The upper level comprises the knowledge layers relating to the organization and the world within which it operates. The organizational layer captures the knowledge processes within the organization, and the external world layer captures those outside it. A companion paper at this meeting analyzes some of the knowledge acquisition requirements for such a system (Sharp 1989).

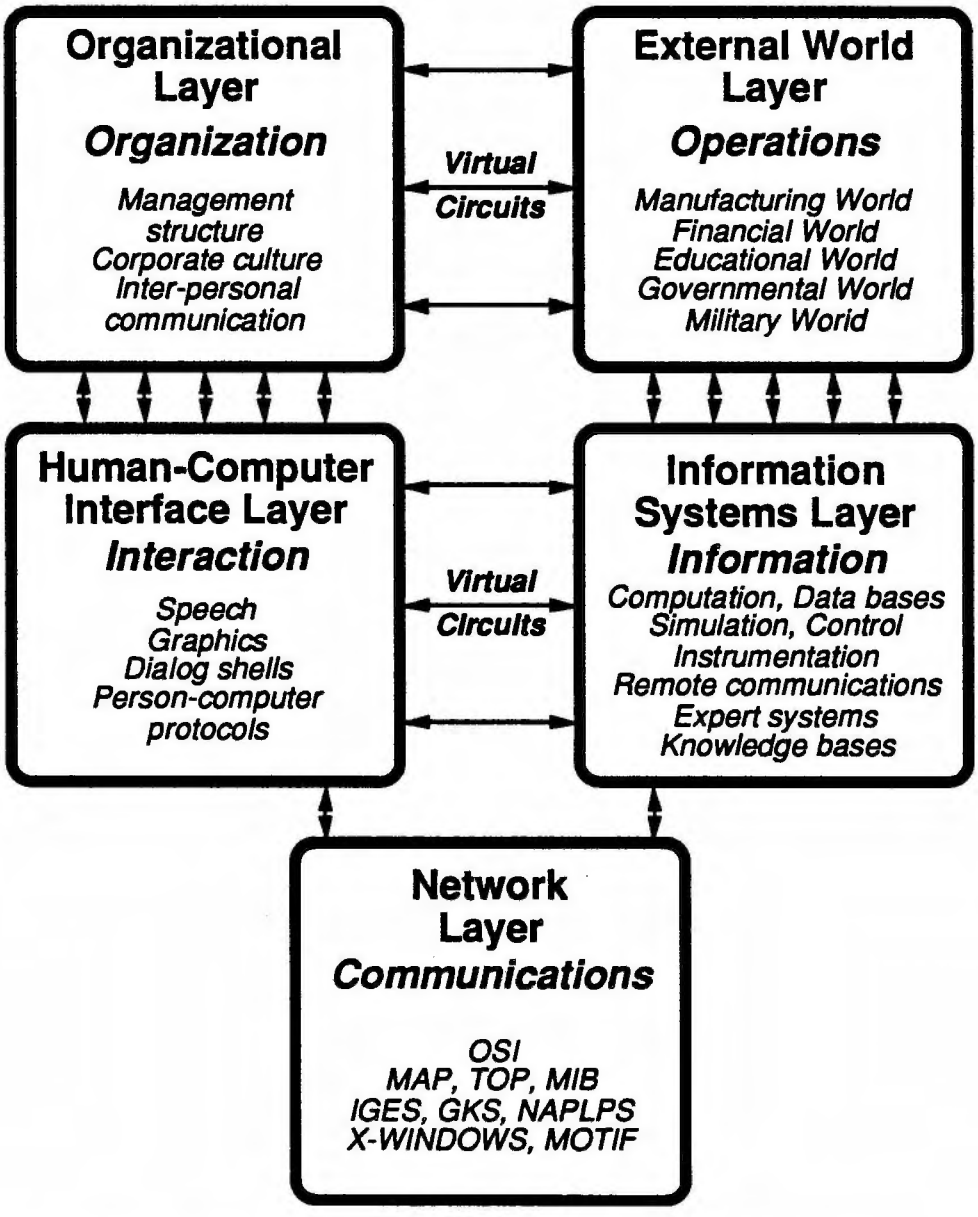


Fig.6 Industrial knowledge support system integrating heterogeneous systems through a hierarchical protocol

CONCLUSIONS

It is difficult to know in writing a paper such as this whether one is helping to expedite system development by defining the overall problem, or whether the magnitude of what is defined is such as to undermine any confidence we may have had that the problems are tractable. It is sometimes better to go into a system development without analyzing requirements and potential problems in too great a detail—this is the way in which breakthroughs often occur. However, as a field matures the capability to produce quality systems consistently that meet requirements, including budgets, schedules and reliability, comes to depend on overt, clearly specified knowledge and well-defined prototypical architectures. The field of knowledge-based systems may become less exciting as this happens but its capability to fulfil its potential will greatly increase.

In terms of the slogans that arise out of the working groups at these meetings, this paper clearly has two, *social context* and *open systems*. Although the first relates to knowledge processes and the second to implementation technology, the two are also closely inter-related since it is the essential openness of an evolving society that generates the need for the openness in its supportive technology. Ultimately there is only one system that we are part of, that we are building, and that is building us.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. I am grateful to many colleagues for discussions over the years that have influenced the research described in this paper, in particular John Boose, Jeff Bradshaw, Gordon Pask, Mildred Shaw, Alain Rappaport, Joan Vickers, Brian Woodward and other colleagues at the Knowledge Acquisition Workshops world-wide. I am particularly grateful to Mildred Shaw for her critical comments on this manuscript.

REFERENCES

Barrett, E., Ed. (1989). **The Society of Text: Hypertext, Hypermedia, and the Social Construction of Information.** Cambridge, Massachusetts: MIT Press.

Beninger, J.R. (1986). **The Control Revolution: Technological and Economic Origins of the Information Society.** Harvard: University Press.

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

Boose, J H. (1989) A survey of knowledge acquisition techniques and tools. **Knowledge Acquisition 1** (1), 39-58 (March).

Boose, J.H. & Gaines, B.R., Eds. (1988). **Knowledge Acquisition Tools for Expert Systems.** London: Academic 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.

Breuker, J. & Wielinga, B. (1987) Use of models in the interpretation of verbal data. Kidd, A., Ed. **Knowledge Elicitation for Expert Systems: A Practical Handbook.** pp.17-44. Plenum.

Buchanan, B.G. & Shortliffe, E.H., Eds. (1984) **Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.** Reading, Massachusetts: Addison-Wesley.

Cercone, N. & McCalla, G., Eds. (1987) **The Knowledge Frontier: Essays in the Representation of Knowledge.** New York: Springer.

Chandrasekaran, B (1988) Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. **The Knowledge Engineering Review, 3**(3), 183-211.

Clancey, W.J. (1987a) **Knowledge-Based Tutoring: The GUIDON Program.** Cambridge, Massachusetts: MIT Press.

Clancey, W. J. (1987b). Intelligent tutoring systems: A tutorial survey. Van Lamsweerde, A. & Dufour, P. Eds. pp. 39-78. **Current Issues in Expert Systems.** London: Academic Press.

Clancey, W. J. (1987c). From GUIDON to NEOMYCIN and HERACLES in twenty short lessons. Van Lamsweerde, A. & Dufour, P. Eds. pp. 79-123. **Current Issues in Expert Systems.** London: Academic Press.

Clancey, W.J. & Shortliffe, E.H., Eds. (1984) **Readings in Medical Artificial Intelligence.** Reading, Massachusetts: Addison-Wesley.

Cullingford, R.E. (1986). **Natural Language Processing: A Knowledge Engineering Approach.** Totowa, New Jersey: Rowman & Littlefield.

Davis, R. (1982). TEIRESIAS: Applications of meta-level knowledge. Davis, R. & Lenat, D.B. **Knowledge-Based Systems in Artificial Intelligence.** pp.229-461. New York: McGraw-Hill.

Durkheim, E. (1893) **The Division of Labor in Society.** New York: Free Press.

Evans, D.A. & Patel, V.L., Eds. (1989). **Cognitive Science in Medicine: Biomedical Modeling.** Cambridge, Massachusetts: MIT Press.

Gaines, B.R. (1986a). Sixth generation computing: a conspectus of the Japanese proposals. **ACM SIGART Newsletter,** No.95, 39-44 (January).

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). 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. (1987c) 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). Organizational integration: modeling technology and management. Zunde, P., Ed. **Empirical Foundations of Information and Software Sciences II.** New York: Plenum Press.

Gaines, B.R. (1988c). Positive feedback processes underlying the formation of expertise. **IEEE Transactions on Systems, Man & Cybernetics,** SMC-18(6), 1016-1020 (November).

Gaines, B.R. (1989a) Social and cognitive processes in knowledge acquisition. **Knowledge Acquisition 1** (1), 39-58 (March).

Gaines, B.R. & Boose, J.H., Eds. (1988). **Knowledge Acquisition for Knowledge-Based Systems.** London: Academic Press.

Gaines, B.R., Rappaport, A. & Shaw, M.L.G. (1989). Integration of acquisition and performance. Boose, J.H. & Gaines, B.R., Eds. **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.25-1-25-20. Banff (October).

Gaines, B.R. & Shaw, M.L.G. (1986). A learning model for forecasting the future of information technology. **Future Computing Systems, 1**(1), 31-69.

Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. pp.47-66.

Gaines, B.R. & Vickers, J.N. (1988). Design considerations for hypermedia systems. **Microcomputers for Information Management**, 5(1), 1-27 (March).
Goodman, D. (1987). **The Complete HyperCard Handbook.** New York: Bantam.
Greif, I., Ed. (1988). **Computer Support of Cooperative Work: A Book of Readings.** San Mateo, California: Morgan Kaufmann.
Hawkins, D. (1983). An analysis of expert thinking. **International Journal of Man-Machine Studies, 18**(1), 1-47 (January).
Heim, M. (1987). **Electric Language: A Philosophical Study of Word Processing.** New Haven, Connecticut: Yale University Press.
Hiltz, S.R., Ed. (1988). **Online Communities: A Case Study of the Office of the Future.** Norwood, New Jersey: Ablex.
Hobbs, J.R. & Moore, R.C. (Eds.) (1985) **Formal Theories of the Commonsense World.** Norwood, New Jersey: Ablex.
Innis, H.A. (1972) **Empire and Communications.** Toronto: University of Toronto Press.
Keen, P.G.W. & Morton, M.S.S. (1978) **Decision Support Systems: An Organizational Perspective.** Reading, Massachusetts: Addison-Wesley.
Kitto, C.M. & Boose, J.H. (1987) Selecting knowledge acquisition tools and strategies based on application characteristics. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Second AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.12-0-12-11. Banff (October).
Kolodner, J., Ed. (1988). **Proceedings Case-Based Reasoning Workshop.** San Mateo, California: Morgan Kaufmann.
Lehrberger, J. & Bourbeau, L. (1988). **Machine Translation: Linguistic Characteristics of MT Systems and General Methodology of Evaluation.** Amsterdam: John Benjamins.
McDermott, J. (1988) Preliminary steps toward a taxonomy of problem solving methods. Marcus, S., Ed. **Automating Knowledge Acquisition for Expert Systems.** pp. 225-266. Kluwer: Boston.
Meyrowitz, J. (1985) **No Sense of Place: The Impact of Electronic Media on Social Behavior.** New York: Oxford University Press.
Michalski, R.S., Carbonell, J.G. & Mitchell, T.M., Eds. (1986). **Machine Learning: An Artificial Intelligence Approach Volume II.** Los Altos, California: Morgan Kaufmann.
Musen, M. (1989). **Automated Generation of Model-Based Knowledge Acquisition Tools.** London: Pitman.
Oddy, R.N., Robertson, S.E., van Rijsbergen, C.J. & Williams, P.W. (Eds.) (1981). **Information Retrieval Research.** London: Butterworths.
Popper, K.R. (1968). Epistemology without a knowing subject. Van Rootselaar, B., Ed. **Logic, Methodology and Philosophy of Science III.** pp. 333-373. Amsterdam: North-Holland
Rappaport, A. (1987) Multiple-problem subspaces in the knowledge design process. **International Journal of Man-Machine Studies 26**(4), 435-452 (April).
Roberts, R.M. (1989). **Serendipity: Accidental Discoveries in Science.** New York: John Wiley.
Schön, D.A. (1983). **The Reflective Practitioner.** New York: Basic Books.
Sergot, M. (1988). Representing legislation as logic programs. Hayes, J.E., Michie, D. & Richards, J., Eds.Macine Intelligence 11. pp.209-260. Oxford: Clarendon Press.
Sharp, M. (1989). Knowledge acquisition for advice systems: FUSION, a network cognitive support system. **Proceedings of the Fourth AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** Banff (October).
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.** pp.30-1-30-19, Banff (November).

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

Shortliffe, E.H. (1976). **Computer-Based Medical Consultations: MYCIN**. New York: Elsevier.

Shu, N.C., Ed. (1988). **Visual Programming.** New York: Van Nostrand Reinhold.

Slonin, J., Schonbach, A., Bauer, M.A., MacRae, L.J. & Thomas, K.A. (1987). **Building an Open System.** New York: Van Nostrand Reinhold.

Smith, D.C. (1977). **Pygmalion.** Basel: Birkhauser.

STA (1985). **Promotion of R&D on Electronics and Information Systems That May Complement or Substitute for Human Intelligence.** Tokyo: Science and Technology Agency.

Steels, L. (1986). Second generation expert systems. Bramer, M.A., Ed. **Research and Development in Expert Systems III.** pp.175-183. Cambridge: University Press.

Swartout, W. R. (1986). Knowledge needed for expert system explanation. **Future Computing Systems, 1**(2), 99-114.

Trigg, R. (1983). A Network-based Approach to Text Handling for the Online Scientific Community. PhD Thesis, University of Maryland (University Microfilms 8429934).

Trigg, R., Suchman, L. & Halasz, F. (1986). Supporting collaboration in NoteCards. **Proceedings of Conference on Computer Supported Cooperative Work**. Austin, Texas: MCC (December).

Van Lehn, K. (1985). Theory reform caused by an argumentation tool. **Xerox PARC Technical Report ISL-11**, 1-15. Palo Alto, California: Xerox Palo Alto Research Center (July).

Vervest, P.H.M. (1988). **Innovations in Electronic Mail.** Amsterdam: North-Holland.

Vliet, J.C., Ed. (1988). **Document Manipulation and Typography**. Cambridge, UK: Cambridge University Press.

Wenger, E. (1987). **Artificial Intelligence and Tutoring Systems.** Los Altos, California: Morgan Kaufmann.

Wojciechowski, J.A. (1986). Computing systems in an ecology of knowledge perspective **Future Computing Systems, 1**(4), 333-352.

Yankelovich, N., Landow, G. & Heywood, P. (1987) Designing hypermedia "ideabases"—the Intermedia experience. **IRIS Technical Report 87-4.** Providence, Rhode Island: Brown University.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A HETEROGENEOUS KNOWLEDGE SUPPORT SYSTEM

Brian R Gaines[1], Alain T Rappaport[2] & Mildred L G Shaw[1]

Knowledge Science Institute[1]
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada T2N 1N4

Neuron Data[2]
444 High Street
Palo Alto
CA94301, USA

ABSTRACT

A knowledge support system is presented that combines hypermedia, knowledge acquisition, expert system shell and database systems to provide an integrated end-user environment. The integration is achieved through inter-application communication protocols operating between four programs designed independently for stand-alone use. The system supports a wide variety of forms and representations of knowledge from the highly informal to the highly computational. It is extremely non-modal and allows knowledge acquisition and knowledge consultation to be combined in a variety of ways. In particular, the results of a consultation may be used to create a new case that may be added to the case base of the acquisition tool which may then be re-analyzed, possibly resulting in changed advice.

INTRODUCTION

This paper presents the architecture and operation of a knowledge support system providing a diversity of knowledge representations, knowledge acquisition paths and validation procedures. The architecture is essentially heterogeneous, based on the loose coupling of four programs originally written as independent stand-alone systems. A uniform and consistent end-user interface is maintained through the commonality and consistent use of the Macintosh window, control and image structures in all systems. Extreme non-modality is supported in that users can switch between tasks and applications freely, either within the integrated application or through the underlying operating system. Multiple partial elicitations, consultations and edits can be undertaken in parallel and left indefinitely in intermediate states. Interlocks are provided to ensure that consistency is maintained in the underlying distributed, heterogeneous knowledge base. The knowledge support system can be used in a highly focused, goal-directed manner for knowledge consultation, or in a highly exploratory, serendipic manner for knowledge creation, or in combination so that application and enhancement are mutually supportive activities.

Any technology carries with it the danger of the Procrustean bed, that we shape our tasks to the available tools and, in so doing, trivialize them rather than address them. When we bring the automata-theoretic precision of computer technology to bear upon knowledge processes, we may filter from the knowledge only that which is computational and, in so doing, lose the essence of what may be known. Fortunately, computer technology has been applied to so many layers of our social activities that links exist to many of the existing media through which we express knowledge, many of which are not truly computational in that their semantics are, as yet, not subject to computer interpretation. Sounds, images and text can be digitized or placed under computer control, and access to them can be made part of a knowledge-based system. Even the semantics of appropriate access do not need to be subject to computer interpretation in that the human user of the system can decide to do so completely outside any computational context.

These considerations extend beyond computer technology to media and discourse processes involving no computer interaction. Our objective in this development has been to recognize that knowledge support systems are part of the technological infrastructure facilitating improved quality and productivity in the knowledge processes of society. Figure 1 shows a model of the knowledge acquisition process that we have previously described in some detail (Rappaport & Gaines 1988). Our systems development is targeted on providing support for as many of the variety of processes shown as is possible with current technology. A major part of the contribution is the computational formalization of knowledge so that it becomes operational, but, to be effective and acceptable, this has to be linked to the rich infrastructure of informal and semi-formal knowledge that affects human behavior even though it not computer-interpretable.

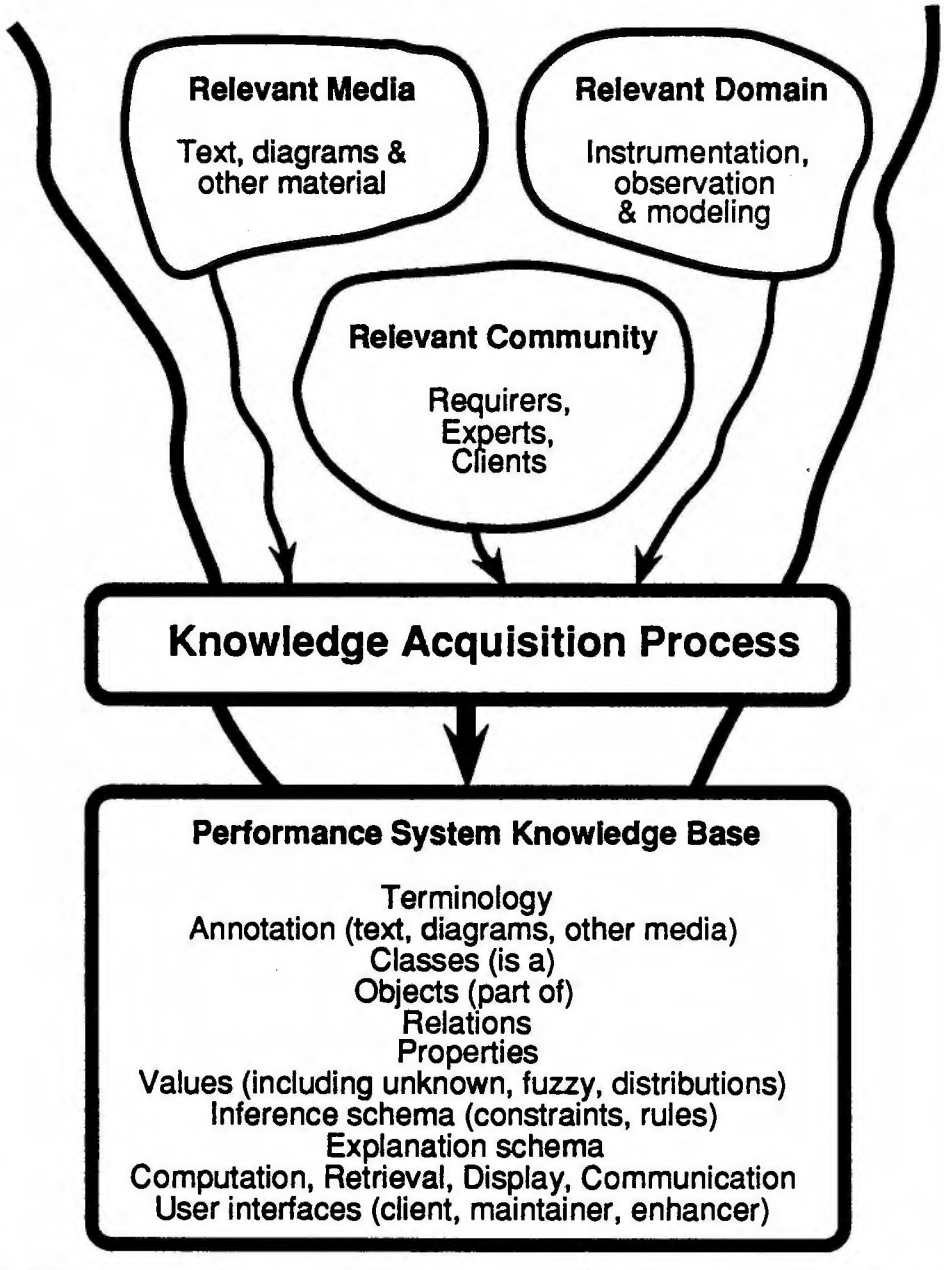


Fig.1 The knowledge acquisition process for knowledge-based systems

KNOWLEDGE SUPPORT SYSTEM ARCHITECTURE

We have previously described two of the tools involved, some of their functionality, and some of the major integration problems in detail (Rappaport & Gaines 1988). Figure 2 was presented as our design objective for a comprehensive knowledge support system. We see a reasonable 'knowledge base' as commencing with fairly informal knowledge that has little structure. Structure may be added to part of this in a variety of ways by annotating it, particularly with cross-references and linkages. Part of this structure may be formalized within various generic ontologies. Part of this formalization may be made operational by encoding it in computational form. Shaw and Woodward (1989) have analyzed the cognitive processes underlying this multi-stage knowledge representation.

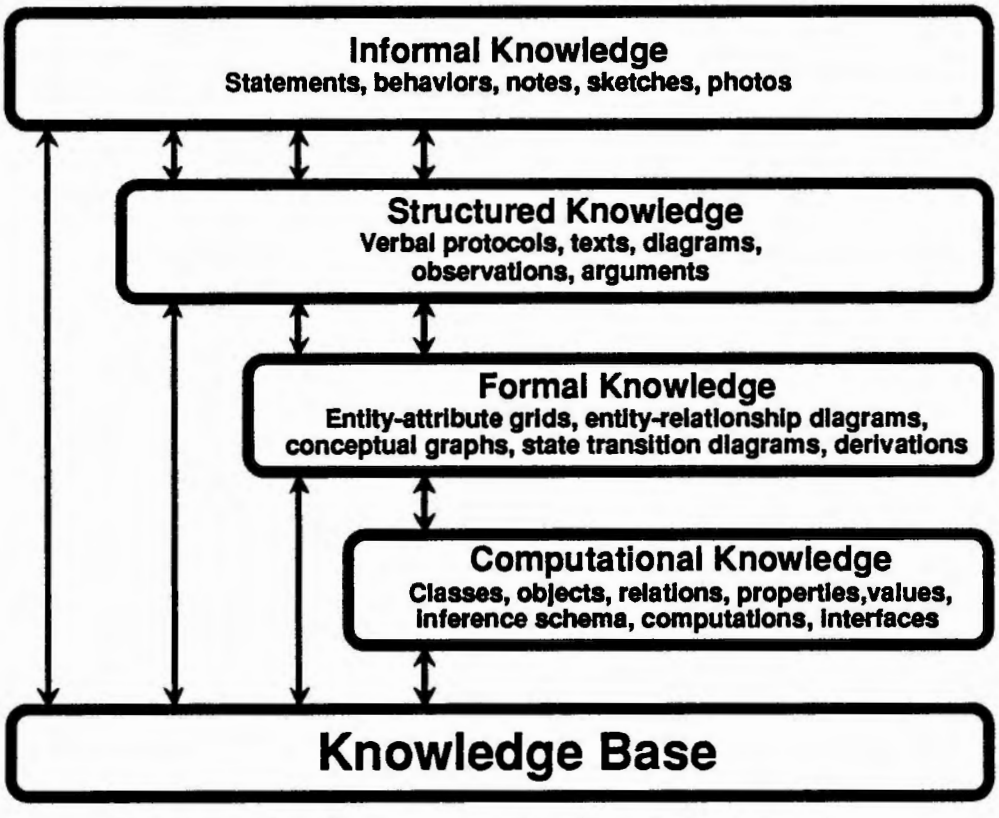


Fig.2 Levels of representation in a knowledge base

In computational terms, it is common to call only the final stage, the computational knowledge, the 'knowledge base.' However, since much of what is captured in the earlier stages is subject to computer storage, if not semantic processing, we see the computer's knowledge base as essentially comprising all these levels of representation. The users' knowledge bases will comprise part of this together with additional material not subject to any form of computer encoding. Even this unencoded material may have computational implications, however, since it may be appropriate to include material about the users' state of knowledge within the knowledge base.

In our system design we have brought together four different existing application packages and integrated them through shared file structures and inter-application communication protocols:

- *HyperCard* (Goodman 1987) is used as a hypermedia system capable of encoding, or controlling access to, some of the informal knowledge and much of the structured knowledge

- *NEXTRA*, an implementation of KSS0 (Gaines 1987b, Gaines & Shaw 1987), is used as a knowledge acquisition system taking the structured knowledge to formal knowledge and encoding it as computational knowledge
- *Oracle* is used as a database holding and processing some parts of the computational knowledge
- *NEXPERT* (Rappaport 1987a,b) is used as an inference engine holding and processing those parts of the computational knowledge most closely associated with expert systems, notably frame structures, inheritance relations and methods.

Figure 3 shows the architecture of the knowledge support system integrating these four application packages. At the bottom of this diagram is the community of users operating in the material, mental and logical worlds (Gaines 1989a). They communicate with the system through an extended workstation. Note that the direct user interface is to only two of the packages. The others are accessed through HyperCard which makes it simpler to provide a uniform and consistent interface. The packages themselves communicate through inter-application communication protocols driving well-defined script languages controlling their functionality. At the top of the diagram, the knowledge base is distributed across the file structures of the four application packages.

This architecture is highly generic and extendable through networked heterogeneous integration (Gaines 1988b, Sharp & Gaines 1989). For example, the 'internal communications' layer could be through local or remote networks, the servers could be on different platforms, and other application servers could be incorporated, such as a simulation package.

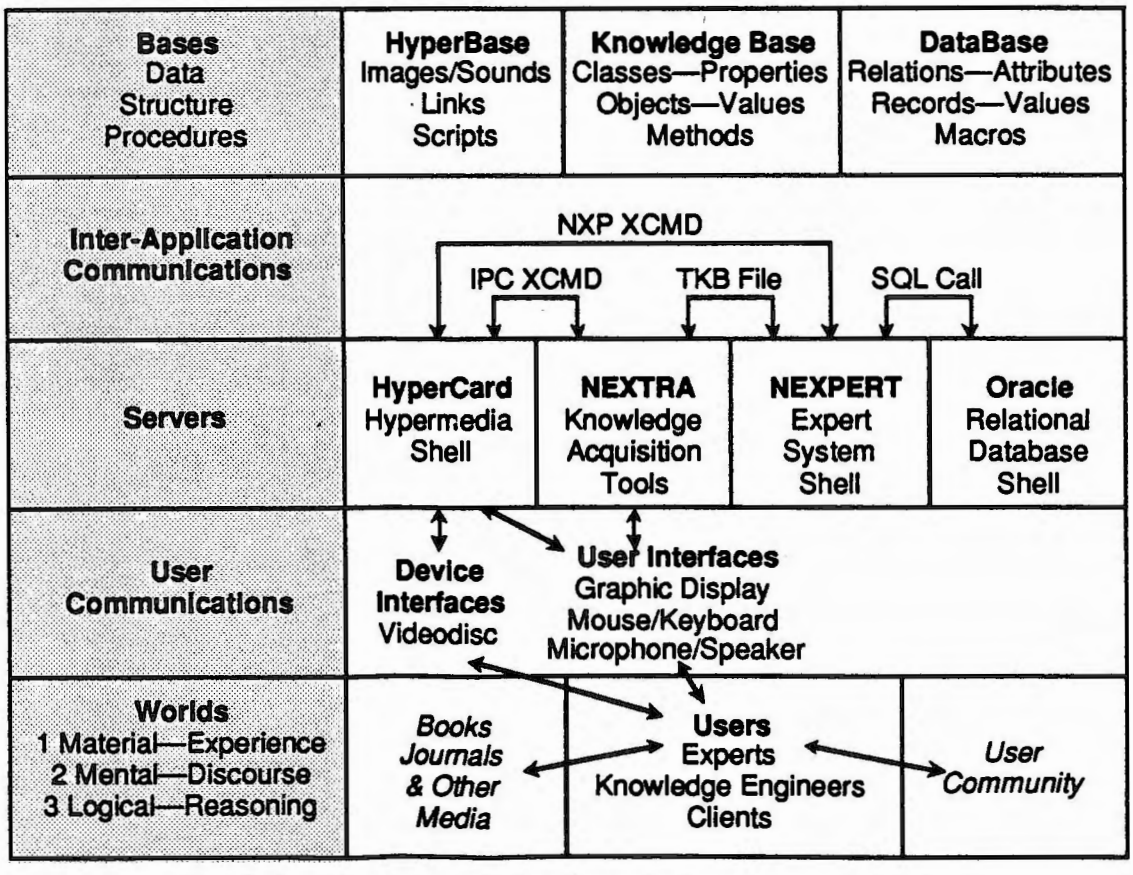


Fig.3 Architecture of the knowledge support system

The primary technical problem in implementing the system of Figure 3 is the maintenance of consistency within the distributed knowledge base including views of it presented to the user, such as edit windows and conceptual clusters. This problem is exacerbated by a design requirement for extremely non-modal operation so that a user may have several editing activities underway and freely move between them. Figure 4 illustrates some of the complexities of the problem as we have previously discussed it (Rappaport & Gaines 1988): if representation B is derived from A then it is clear that editing A may result in the need to propagate changes to B; however, it is less clear how to cope with changes to B that may, or may not, be consistent with the original source A.

The approach we have adopted to this solution currently is to keep track of the state of all data structures and their relations and tag them with version numbers so that the information, for example, that version b of B derived by process X from version a of A is continuously available. Since even the names of structures can be edited, unique accession numbers are generated as structures are created and used for inter-application linkages. Updates are done on a lazy basis and are activity dependent. For example, the knowledge base created by the acquisition tool for use by the expert system shell is automatically updated if necessary when a consultation is initiated with the shell, but not if acquisition is undertaken, as it may be, during the process of consultation.

The most difficult design decisions relating to such updating are those concerned with data in windows for user interaction and visualization. We do not know where the user is looking unless we force an action in a window, so that lazy updating is inadequate. However, continuous updating may be both time-consuming and hinder user interaction, for example if the user is actually attempting to determine the effect of a change in a case upon the results of conceptual clustering or constraint induction. We are experimenting with various forms of user interaction to determine what functionality makes the system most effectively usable.

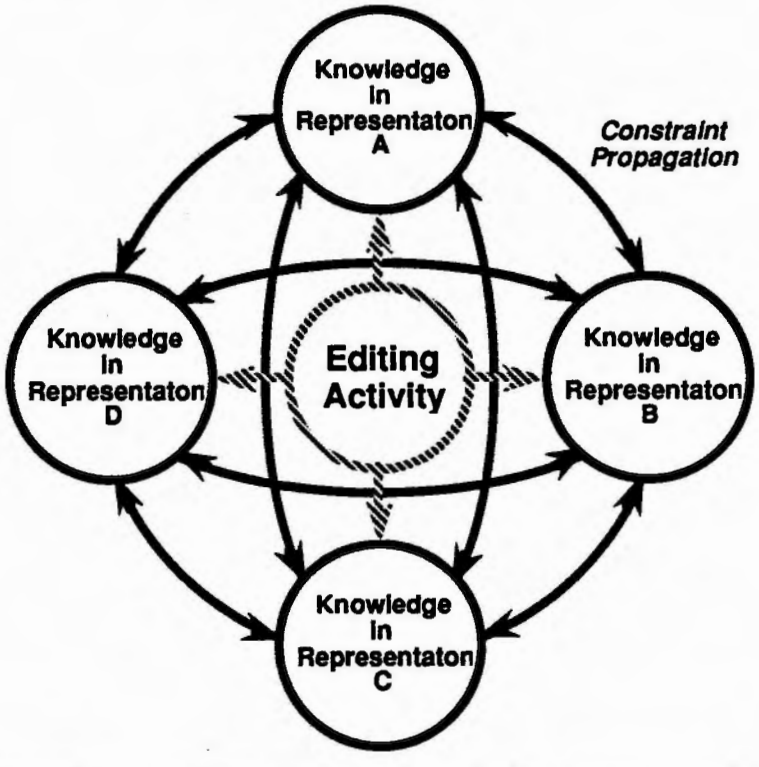


Fig.4 Mutual constraints between knowledge structures in editing

KNOWLEDGE SUPPORT SYSTEM IN ACTION

This section presents in detail the operation of the knowledge support system whose architecture is shown in Figure 3. The starting point for a user is an initial hypertext stack with two backgrounds and associated scripts, three cards, and a stack script. The user can add additional backgrounds and cards for annotation and other purposes. Figure 5 shows this basic stack structure.

The "control" background and cards are used to initiate major activities in the expert system shell such as loading knowledge bases and consulting them, and in the knowledge acquisition system such as conceptual clustering, rule induction and knowledge base creation. Two cards are provided, one blank and subject to user embellishment to give the stack and identity, and the other containing a scrolling log field through which all activities may be monitored. Either of these, or additional user-defined cards, may be used for primary control.

The "data" background and cards are used to provide basic annotation of attributes and entities in the dataset associated with the stack. One card is provided that asks the user whether a new card should be created, and the other cards are created dynamically as new entities and attributes are added. Each card contains two scrolling fields: one set up automatically with the possible values of attributes for attribute cards, and with the actual values of attributes for entity cards; and the other made available to the user for annotation. The background also provides buttons for editing data, adding an entity to the knowledge acquisition dataset, and passing control back to the knowledge acquisition tool. Space is left for user embellishment with graphics and buttons linking to extended annotation.

<table>
<tr><th></th><th>Control</th><th>Data</th><th>Extensions</th></tr>
<tr><td>Backgrounds</td><td>System-defined background for primary control</td><td>System-defined background for primary data</td><td>User-defined backgrounds for annotation structures</td></tr>
<tr><td rowspan="2">Cards</td><td>System-defined initial card subject to user-embellishment</td><td>System-defined "Add a new item?" card</td><td rowspan="2">User-defined cards for annotation</td></tr>
<tr><td>System-defined log card</td><td>System-generated cards for entities & attributes subject to user-embellishment</td></tr>
<tr><td>Background Scripts</td><td>Load knowledge base
Start consultation</td><td>Answer NEXPERT
Edit cases
Post cases to NEXTRA</td><td>User-defined scripts</td></tr>
<tr><td>Stack Script</td><td colspan="3">Inter-program protocols</td></tr>
<tr><td rowspan="2">Code Segments</td><td rowspan="2">Hypertalk interpreter</td><td>NXP XCMD</td><td>IPC XCMD</td></tr>
<tr><td>NEXPERT dynamic library</td><td>Inter-program communication</td></tr>
</table>

Fig.5 Stack structure

The "extensions" background and cards are freely created by the user for purposes such as additional annotation of entities and attributes. Since the complete power of the script languages of all four application packages is available in these extensions, it is possible to build major knowledge acquisition tools within the stack, or called by inter-program communication from the stack. In particular, a knowledge engineer can create as domain-specific an environment as is appropriate.

Figure 6 shows the initial control card of a stack developed as part of knowledge engineering for a domain concerning national science and technology policies. The user has added a world map, a button linking to an additional control card showing world demographics, and buttons linking to the cards created for nations which are the chosen entities. These links are particularly simple because the card creation script names the cards by the entity names, and the stack script contains a generic button script that goes to the card named on the button if the button itself has no script.

The other buttons on this card are provided by the system to control: the creation of a knowledge base from an entity-attribute dataset, its loading and consultation through the expert system shell; and the operation of the knowledge acquisition tool in processing the dataset. Control may be transferred explicitly to the user interface of the acquisition tool, and control is then transferred back through context-sensitive pop-up menus in that tool. The user interfaces of the expert system and database tools are through the hypertext tool so that such explicit transfer of control does not occur.

The inter-program communication protocol allows the acquisition tool also to be used as a library of callable routines rather a separate application with its own user interface. However, HyperCard currently does not provide the variable size colored windows necessary to the full functionality of the acquisition application.

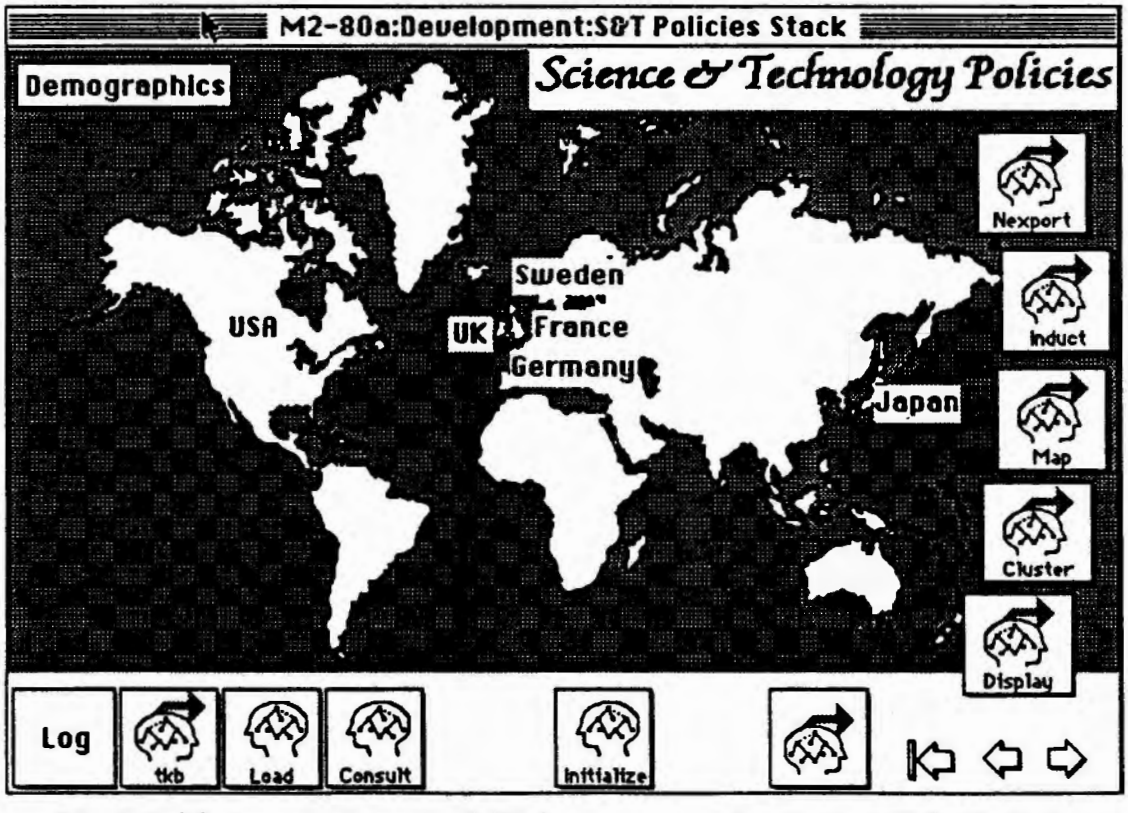


Fig.6 Initial control card of "Science and technology policies" stack

Figure 7 shows the log control card which keeps track of the major activities that have occurred. The knowledge engineer may use this card to keep track of the full complexity of a knowledge acquisition session. The normal question mark indicates user replies to query from the expert system shell during a consultation, and the inverted question mark indicates conclusions drawn. The final action recorded in this log is the adding of the entity resulting from an edited consultation to the dataset used by the acquisition tool.

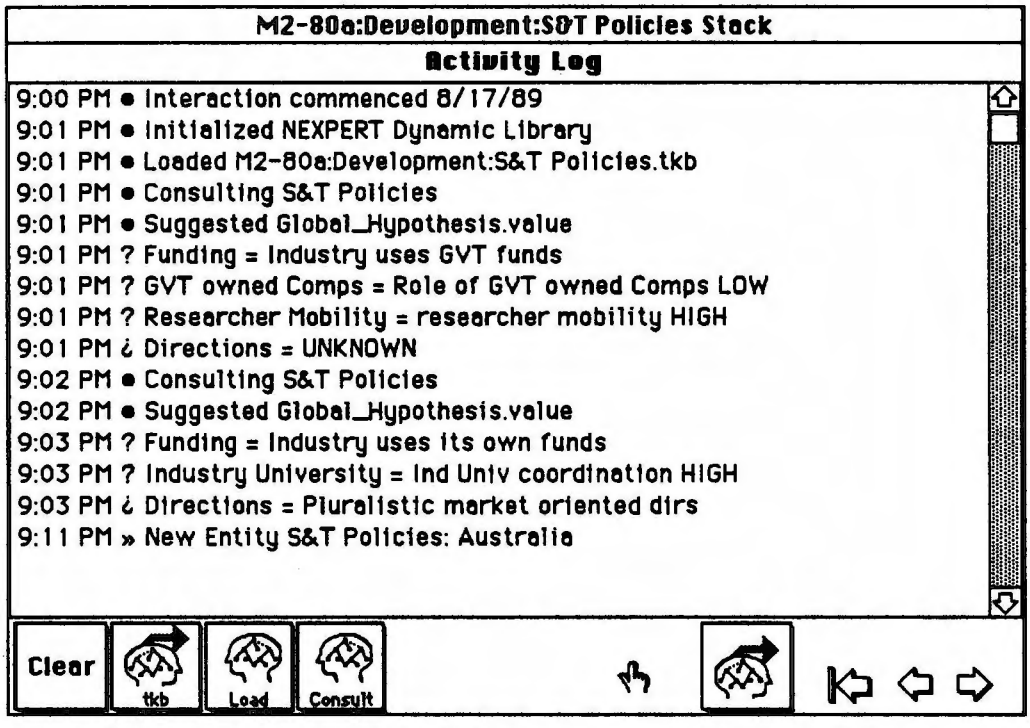


Fig.7 Log control card of "Science and technology policies" stack

Figure 8 shows a data card for the entity "United States." It is created automatically when the stack is accessed from the knowledge acquisition tool with the first request for annotation associated with this entity. The top three fields giving the domain name, entity name and entity values on attributes are created from the knowledge acquisition dataset. The annotation in the scrolling field on the lower left has been added by the user, as has the map on the right.

Figure 9 shows a set of windows forming the user environment at some stage during the acquisition process. For the user only one application is apparent. The two applications keep track of which window is in front and update their data structures appropriately. The user can bring any window to the front, possibly changing the underlying application, without being aware of the interlocks being activated to prevent inconsistencies.

Figure 10 shows how the data card of Figure 8 may be accessed from an interview window used in knowledge elicitation through the knowledge acquisition tool. As the user mouses across the names of structures in the dataset such as entities and attributes, they highlight to show they are selectable and at their extreme left side the cursor changes to a menu shape to indicate that a pop-up menu is available. If the use presses the mouse button the pop-up menu appears as shown in Figure 10, offering a range of actions, one of which is transfer to HyperCard. This option only appears in the menu if HyperCard is running and, if selected, transfers control to the hypertext tool passing information about the object selected, the window and the dataset. This is sufficient for the inter-program communication protocol script to find and transfer control to the appropriate stack and card.

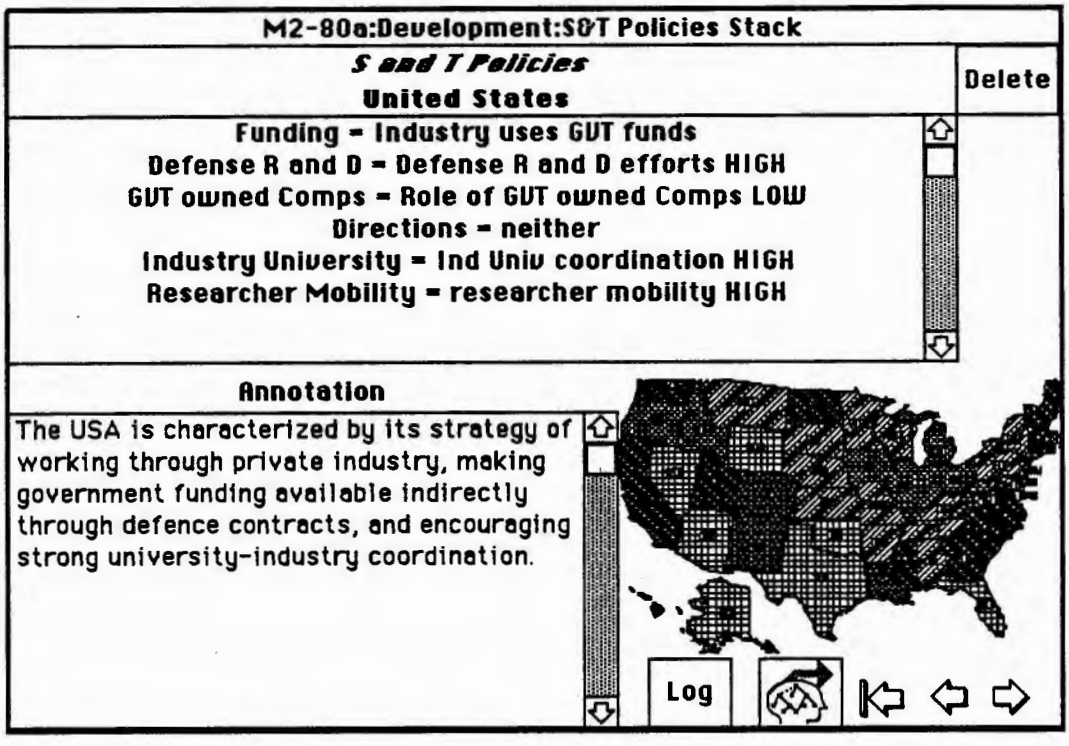


Fig.8 Data card for the entity "United States"

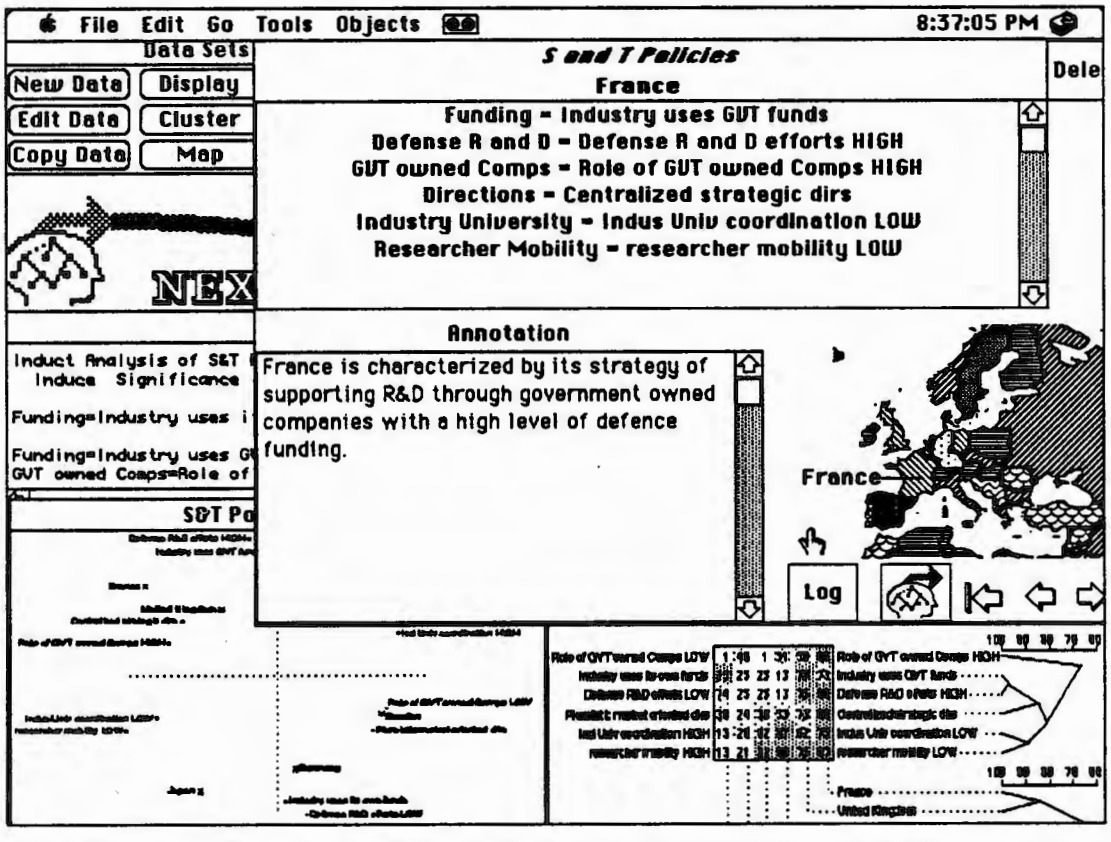


Fig.9 User environment of hypertext and knowledge acquisition windows

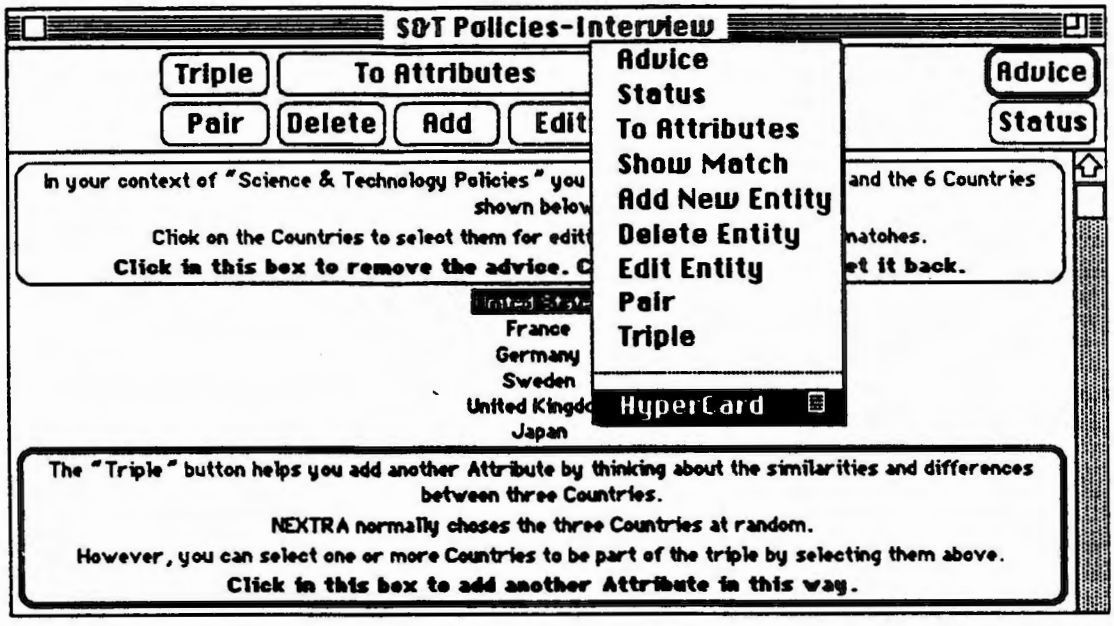


Fig.10 Context-sensitive pop-up menu in the knowledge acquisition tool

The pop-up menus are consistently available in all contexts. Figure 11 shows one being activated by a user browsing a conceptual map of the dataset created by the knowledge acquisition tool. Thus access to the annotation facilities in the hypertext tool is readily available through simple interaction in the knowledge acquisition tool. The expert from whom formal knowledge is being elicited can also enter informal and structured knowledge and link it automatically to the formal knowledge. This can include links to other media accessible through the hypertext tool such as videodiscs and sounds, including voice annotation.

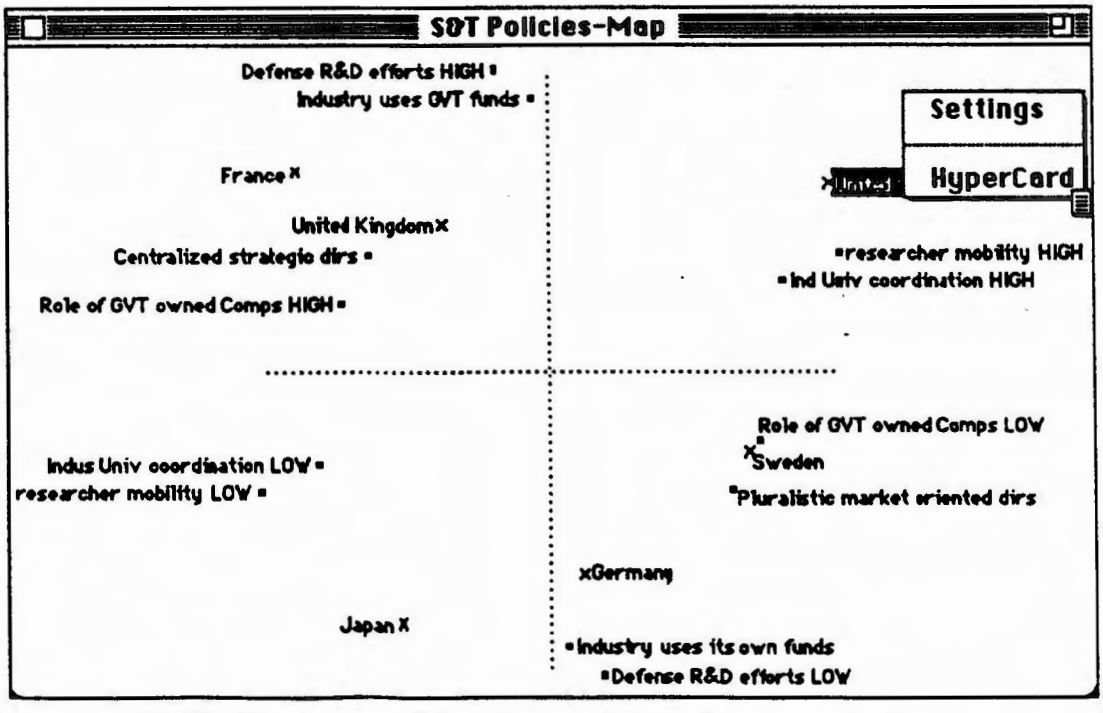


Fig.11 Context-sensitive pop-up menu in conceptual map

The data cards created in the hypertext tool are not only used for annotation. They are also the primary user interface to the expert system shell. The knowledge acquisition tool exports to the expert system shell a knowledge base in which classes, properties, objects, values and methods have been created that correspond to the elicited dataset and the entailments derived from it. This may be consulted through the expert system shell by clicking on the "Consult" icon on the lower left of Figure 6. This leads to the shell checking inference rules in the knowledge base and initiating queries to the user about the values of attributes tested in those rules.

The inter-program communication protocol passes the queries to a script that obtains answers from the user using the appropriate attribute data card as a user interface as shown in Figure 12. The top scrolling field is filled in automatically from the dataset as the set of possible values of the attribute, including the possibility of it not being known. The lower scrolling field is annotation added by the expert at some time, usually when the attribute is elicited. The space on the lower right is left for additional graphics, links, videodisc control, and so on, which may be used for tutorial and explanatory purposes when a question about the attribute is asked.

The upper two fields of the attribute data card are the domain and attribute name. In Figure 12, when the card is being used for a query, these are overlaid with the query. The user answers it by clicking on the appropriate value in the scrolling field below.

When the expert system shell reaches a conclusion it creates a new entity card showing the results of the consultation as shown in Figure 13. The values in the top scrolling field show both the answers to queries and the conclusions reached. The latter are also placed in the annotation field.

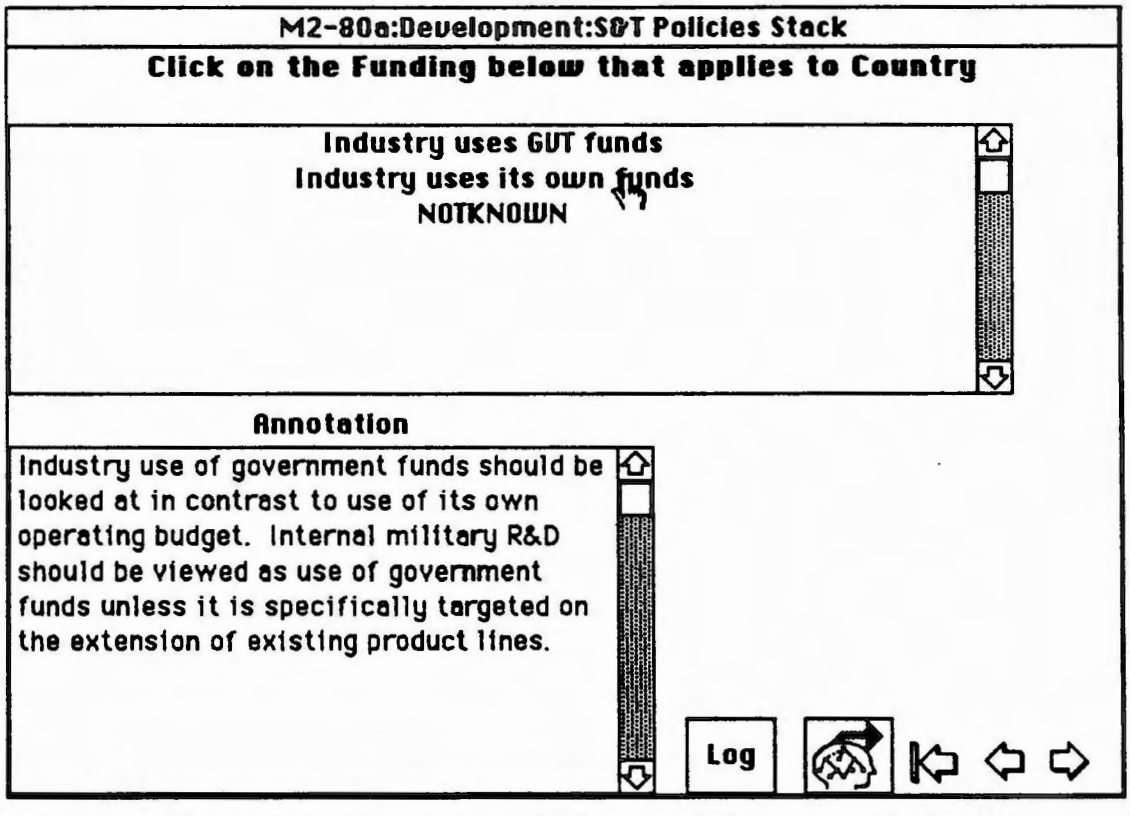


Fig.12 User query through data card in a consultation

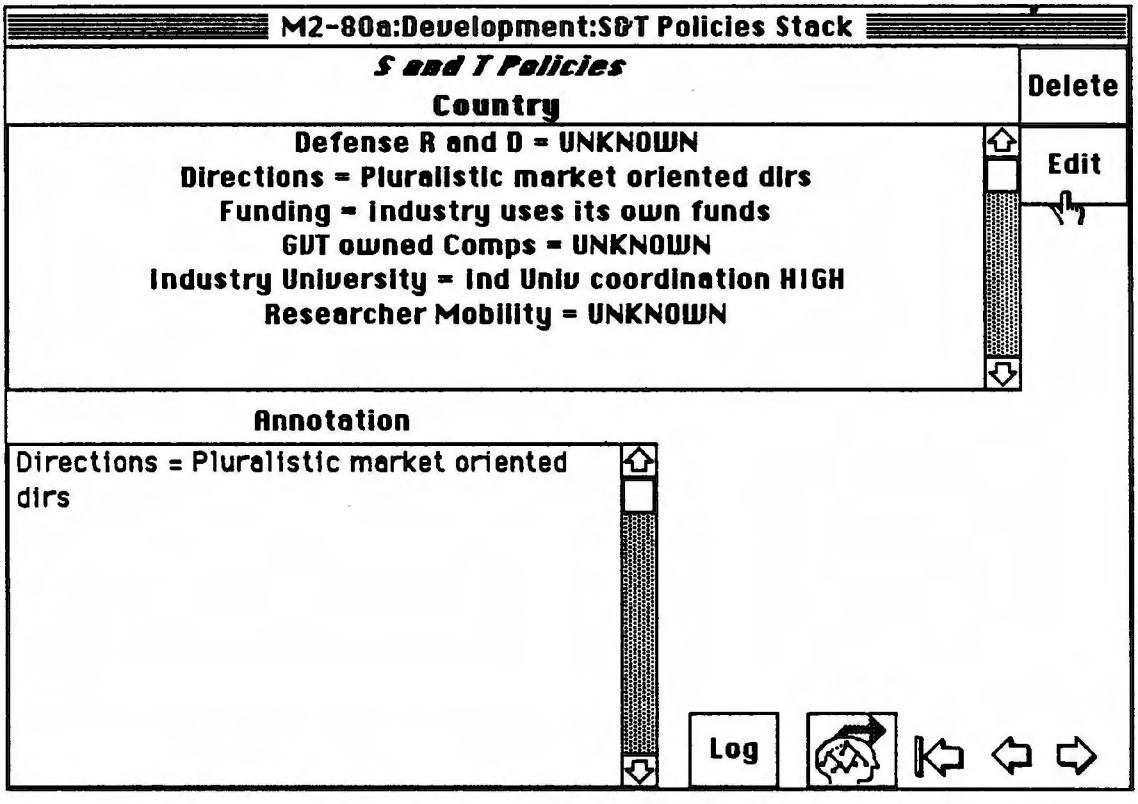


Fig.13 Results of a consultation

One outcome of the consultation, particularly during knowledge acquisition, is that the user is not satisfied with the results and wishes to change them. Clicking on the "Edit" button at the upper right of the card puts the system into an edit mode in which the user can name the test case, fill in values of attributes not requested by the shell during the consultation, and change the conclusion appropriately.

Editing the value of an attribute is carried out by clicking on it in Figure 13 which results in transfer to a query card again using the appropriate attribute data card as shown in Figure 14. Clicking on the required value in Figure 14 transfers control back to the card of Figure 13 with the new value entered.

When the editing is complete a new entity has been created as shown in Figure 15. Clicking on the "Add" icon at the upper right posts this entity to the knowledge acquisition tool where it is added to the dataset. Running entailment induction may result in changed rules in the new knowledge base and hence changed behavior in a consultation with the expert system shell. The effect of the new entity on the conceptual structure may be investigated immediately within the knowledge acquisition tool—for example, Figure 16 shows the resulting conceptual cluster including the new entity.

The non-modal nature of the user interface means that such edits to the dataset have to be done in such a way that no problems arise if the dataset is already open for editing in the knowledge acquisition tool. In particular, edit windows must be automatically updated to show the new data. The inter-program communication protocol will also operate across a network so that the edit windows could be in active use when they are updated. It may be appropriate to ask the other user whether the new entity should be added—the interaction of technical and social protocols for simultaneous editing of the same dataset needs substantial further investigation (Gaines 1986).

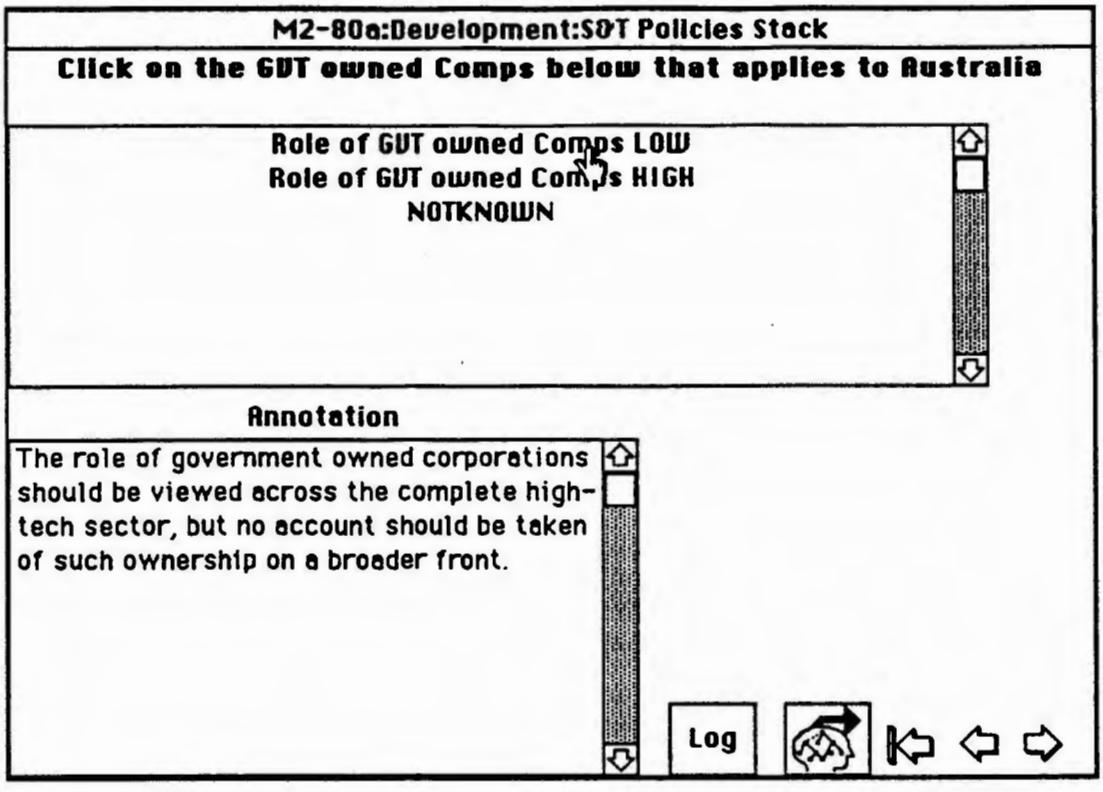


Fig.14 Editing results of a consultation

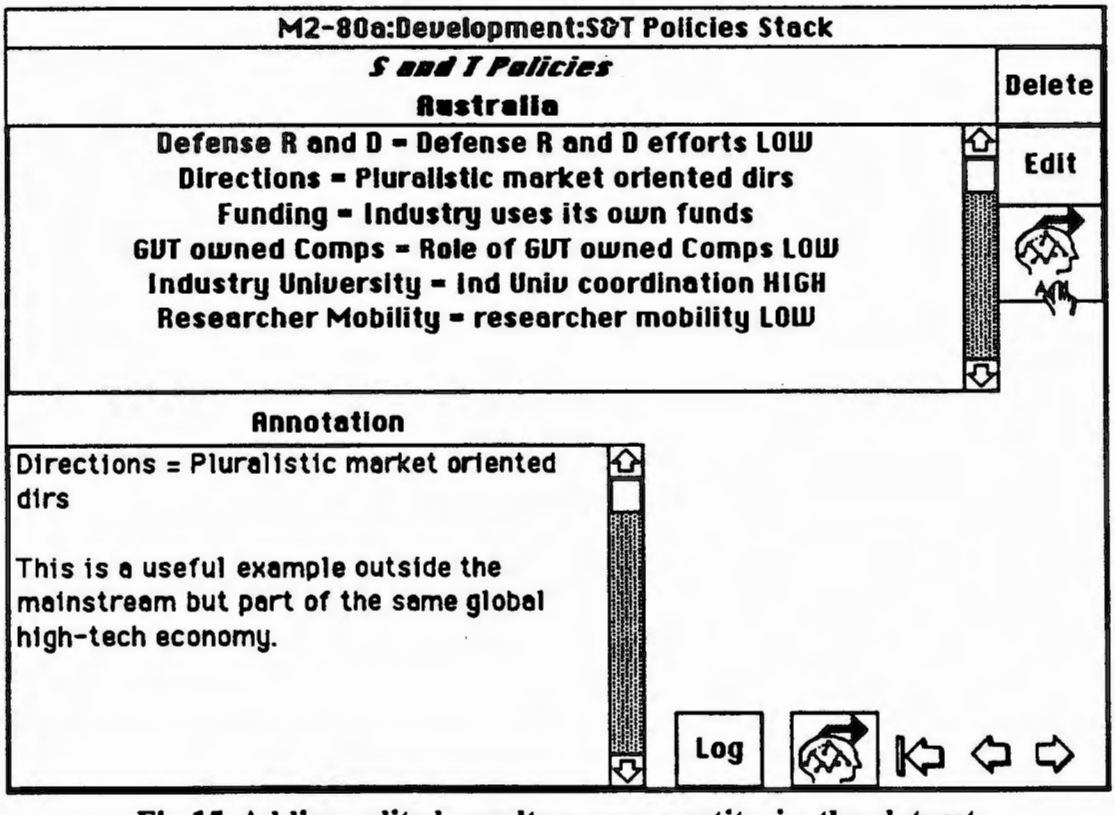


Fig.15 Adding edited results as new entity in the dataset

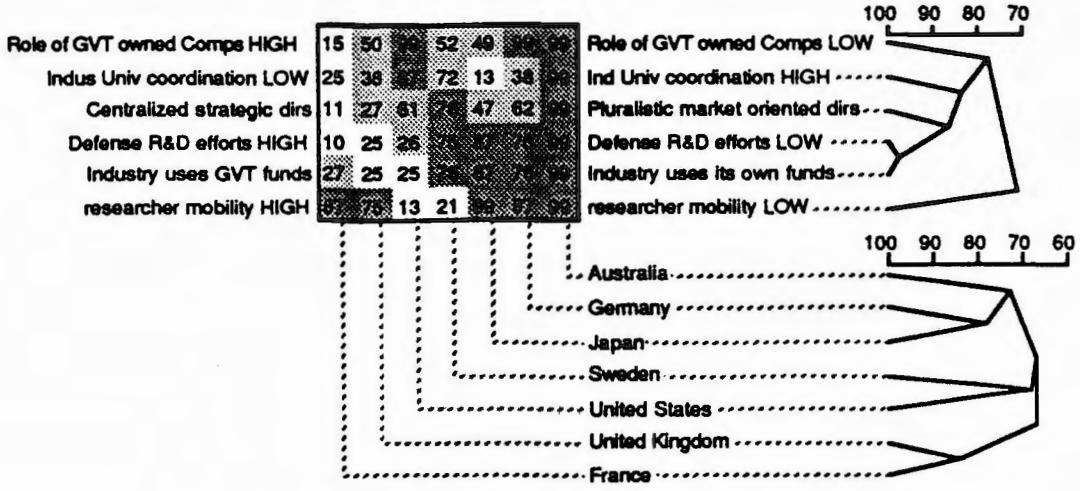


Fig.16 Concept cluster showing new entity added

OVERALL APPLICATION OF THE KNOWLEDGE SUPPORT SYSTEM

Figure 17 shows the modes of knowledge acquisition available in the knowledge support system we have described. We have analyzed previously the many psychological problems that impede knowledge elicitation (Gaines 1987a, Gaines & Boose 1988) and emphasized the need for a wide variety of approaches and technologies to be made available. The system described provides many qualitatively distinct alternatives for elicitation.

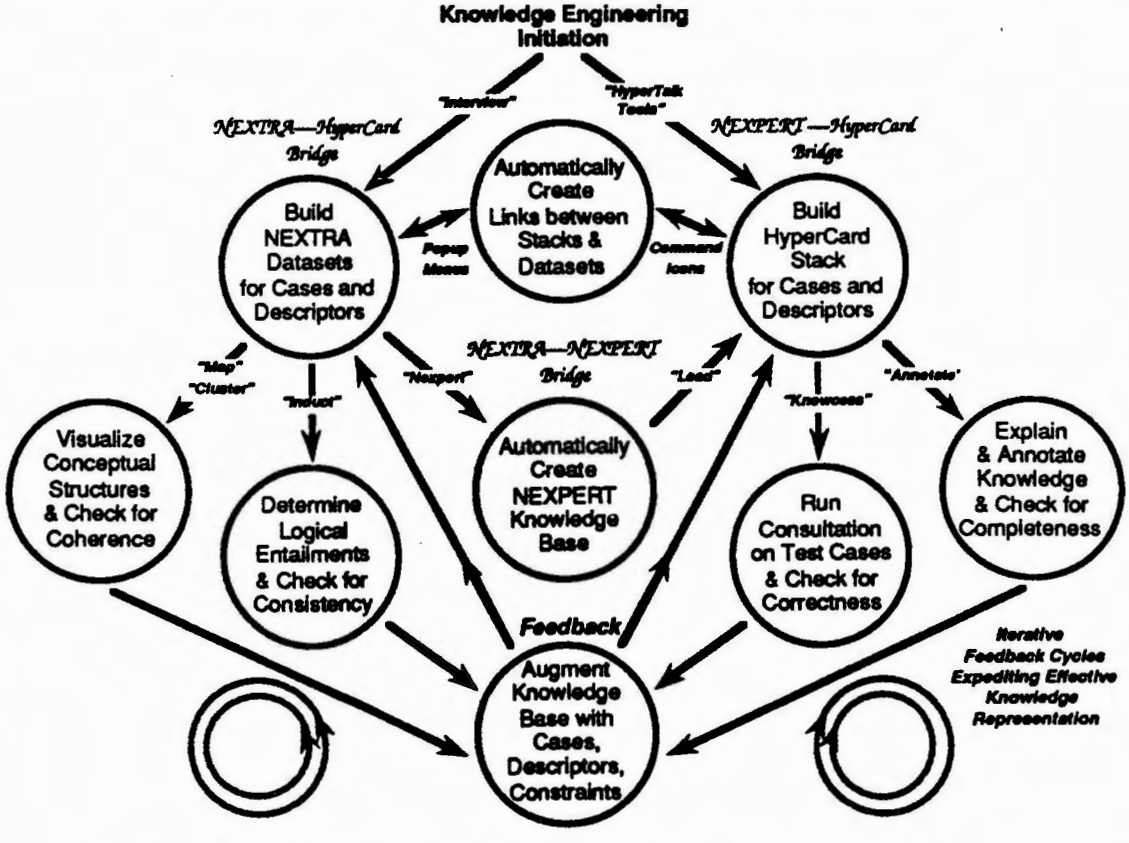


Fig.17 Modes of knowledge acquisition available

The knowledge engineering activity can be initiated either within the knowledge acquisition tool or within the hypertext tool. The expert can commence with structured knowledge elicitation and annotate it, or commence with an informal notebook and structure it. The system will automatically link the stacks and datasets, no matter how generated, and the expert can add further linkages to enrich the knowledge base. The system also automatically makes the computational part of the knowledge base mutually available to the acquisition tool and expert system shell so that consultations are available at any time.

It is interesting to analyze the multiple paths for knowledge elicitation in terms of their impact on the validation of the knowledge elicited:

- The visualization analyses in the acquisition tool allow the conceptual structures elicited from the expert to be examined and checked for *coherence*, that the internal relations between cases, between descriptors, and between cases and descriptors make sense to the expert.
- The logical entailment analyses in the acquisition tool allow the constraints implied in the conceptual structures elicited from the expert to be examined and checked for *consistency*, that they can not lead to contradictory conclusions.
- The consultation facilities in the expert system shell allow the knowledge base derived from the conceptual structures and constraints to be checked for *correctness*, that conclusions acceptable to the expert are generated with test cases.
- The annotation facilities in the hypertext tool allow the computational knowledge base derived from the conceptual structures and constraints to be checked for *completeness*, that what has been made computational adequately encompasses the critical material described less formally.

The terminology above is reminiscent of notions of truth in the philosophy of science and philosophical logic, and the connotations in terms of validation procedures are similar. However, the knowledge elicitation process can not be reduced to formal considerations of truth. Even expertise in a formal domain, such as mathematics, involves relevant knowledge outside the formal domain, such as worthwhile problems, fruitful approaches to unsolved problems, problem-solving techniques, and so on. The notions of truth in formal systems are, however, the end-product of refining validation procedures for proto-truth as shown in Figure 17.

There are also elicitation and validation processes available within the system that are not represented in Figure 17. The inter-expert comparison facilities in the acquisition tool (Shaw & Gaines 1988) allow the grounding of the experts' terminology and its relation to distinctions made to be analyzed. The database tool allows existing sources of information to be used in building the knowledge base, through expert selection and empirical induction rather than expertise transfer (Gaines 1989b). The plurality of sources and processes leading to the creation of an adequate knowledge base is important in allowing some of the variety of knowledge-based systems to be encompassed, and some of the blockages in knowledge elicitation to be bypassed. It increases the managerial load on the knowledge engineer, to make effective use of appropriate tools.

The availability of many modes of elicitation also makes it more difficult to design a comprehensible user interface. We have avoided some problems currently by treating the expert system shell and database tools as libraries callable from scripts in the hypertext tool. In so doing, we have lost access to some significant facilities in the application programming support environments of these tools. Currently, these may be accessed separately but not as part of the integrated system.

EXTENSIONS

The knowledge support system described in this paper, even though it has very powerful functionality, is also completely open in its architecture. The functionality shown is controlled through scripts in the hypertext tool that access the other tools as callable libraries, each having its own script language. The control scripts and the library scripts are easily modified and extended. In particular, new knowledge elicitation and application modules can be added, and new links may be made to other application tools. We see this as the essence of modern information system design, to provide base technology in modular form with an open architecture already delivering powerful functionality, yet extendable to be more general or more specific according to user community requirements.

For example, a knowledge engineer could build a stack in the hypertext tool that was highly targeted to a specific domain area, incorporating structures and vocabulary that were already standard for experts in that domain. Domain specific knowledge acquisition tools have been at the heart of some of the most successful applications of knowledge-based systems (Boose & Gaines 1988, Musen 1989), but it is a major task to create an effective one for a new domain. Customizing existing generic tools to be as domain-specific as required offers a highly productive approach to specialized knowledge acquisition.

The extensions need not be domain-specific but could add elicitation and analysis functionality to the tool at different levels from those already covered, or using different knowledge acquisition paradigms. For example, Woodward's (1988) operationalization of linguistic elicitation of 'implicit knowledge structures' is implemented in HyperCard as his Cognosys tool, and could be incorporated as another starting point for elicitation standing between the unstructured hypertext tool and structured entity-attribute acquisition tool.

The extensions could also be used for integrating different knowledge islands within a domain. Entity-attribute grid tools (Shaw 1980, Gaines & Shaw 1980, Shaw & Gaines 1980, 1987, Boose 1984, 1986, Boose & Bradshaw 1987) may be seen as being most effective at the class or frame level. They allow a single, coherent knowledge island to be thoroughly explored. This may be all that is required in simple classification and selection tasks. However, in more complex domains, knowledge structures for a number of knowledge islands have to be developed and then integrated through appropriate import-export data linkages and control structures to operate as an effective overall system. The acquisition tool can already cope with multiple knowledge islands as separate entities, and we are developing additional facilities within it to cope with the integration. However, the open architecture described in this paper makes it feasible to program the integration facilities within the hypertext tool, and this may be the appropriate place for them since it is not yet clear that they can be provided in a form that is both usable and domain-independent.

One may continue this line of thinking further and contemplate future system designs that are even more modular than that described in this paper. The acquisition tool is written as a loose control framework calling separate modules for graphic elicitation, visualization, expert comparison, induction, and knowledge base creation. These modules could be separated as distinct libraries and the control framework moved into the hypertext tool. Currently, there are weaknesses in existing hypertext tools, notably lack of variable-size color windows, that make this unattractive, but it may become the most effective architecture as these tools develop.

HyperCard shows the potential for system design with an object-oriented development tool supporting a general-purpose interpretive language, with recourse to foreign function calls in other languages when speed is required, and having a powerful user interface. There are weaknesses in the consistency of the design, the language and the interface, but these can be overcome, and are already outweighed by the system design possibilities.

The next generation of such tools will make major changes to system design practice, particularly for highly interactive, intrinsically heterogeneous systems, such as knowledge support systems. Future tools may be based on the evolution of HyperCard-like tools, or on the combination of more principled systems such as the Smalltalk object-oriented programming language coupled with the X-Windows/Motif networked user interface and inter-application protocols. However the technology is provided, it offers opportunities for system designs and implementations that were virtually impossible a few years ago, yet which are fundamental to our aspirations for knowledge-based systems.

We have to be very careful that the technological limitations of the past have not blinkered us to the opportunities now becoming available. The next generation of knowledge support systems will be limited more by our creative imaginations, than by our available technologies.

CONCLUSIONS

This paper describes the architecture and operation of a knowledge support system that combines hypermedia, knowledge acquisition, expert system shell and database systems to provide an integrated end-user environment. The integration is achieved through inter-application communication protocols operating between four programs designed independently for stand-alone use. The system supports a wide variety of forms and representations of knowledge from the highly informal to the highly computational. It is extremely non-modal and allows knowledge acquisition and knowledge consultation to be combined in a variety of ways. In particular, the results of a consultation may be used to create a new case that may be added to the case base of the acquisition tool which may then be re-analyzed, possibly resulting in changed advice, and supporting the essentially iterative process of knowledge-based system development shown in Figure 1.

We have emphasized the knowledge acquisition phase of application of the system in this paper, yet the integration of performance and acquisition tools blurs the distinction between development and application phases. The possibility of acquisition through performance allows knowledge engineering to be treated as an ongoing process with new case data being added from field experience in the application of the system. This is a step towards what Steels (1986) and Gaines (1988a) have termed "second generation" knowledge-based systems able to learn through their performance. Our current implementation requires extension to incorporate anomalous case detection to draw attention to cases where the knowledge base may be inadequate rather than clearly in error (Gaines 1988c). To fully participate in the knowledge processes shown in Figure 1, it is also important for the system to be able to report local updates to other systems through remote communications. Such extensions present no significant technical problems, but they require a different conception of the role of expert systems than is currently prevalent in user communities.

If there is one overall conclusion to be drawn from this paper, it is the importance for modern open system architectures to achieving the aspirations of the knowledge acquisition and knowledge-based systems development communities. We are now able to make available tools that are powerful and well-integrated in their own right, but where we are also able to say to user communities, "What we provide is useful, but you are not bound by it. If you want to change or extend the system, it is open to you do so." This catches the essence of knowledge processes in society, that each of us stands on the shoulders of countless others in making our own contribution, that we take the tools developed by earlier generations and other disciplines and use them in unintended ways for new purposes. Knowledge support tools have to be open, malleable and extensible if they are to have a meaningful role in the knowledge processes of society.

It is interesting to look back at our own aspirations for knowledge support systems and see if we are still on track and whether they have themselves been extended. We state that (Shaw & Gaines 1986):

1. The knowledge support system should be domain independent.
2. The knowledge support system should be directly applicable by experts without intermediaries.
3. The knowledge support system should be able to access a diversity of knowledge sources including text, interviews with experts, and observations of expert behavior.
4. The knowledge support system should be able to encompass a diversity of perspectives including partial or contradictory input from different experts.
5. The knowledge support system should be able to encompass a diversity of forms of knowledge and relationships between knowledge.
6. The knowledge support system should be able to present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations.
7. Users of the knowledge support system should be able to apply the knowledge in a variety of familiar domains and freely experiment with its implications.
8. The knowledge support system should make provision for validation studies.
9. As much of the operation of the knowledge support system as possible should be founded on well-developed and explicit theories of knowledge acquisition, elicitation and representation.
10. As the overall knowledge support system develops it should converge to an integrated system.

The system described in this paper satisfies all these objectives, with the greatest incremental contribution being to the final objective, of convergence to an integrated system. However, our concept of what such integration means is certainly very much looser and heterogeneous than it was three years ago. We have also extended each of our objectives, for example, the first would now read, "be domain independent at their kernel but customizable to be as domain specific as is useful." Indeed this customization is now a major requirement in its own right since it impacts all the other objectives—direct applicability may not always be achievable without prior customization—the incorporation of theoretical developments may requires extensions to the tools, and so on.

The degree and quality of satisfaction of each of these objectives is clearly open to major improvement—and this will always be so—knowledge is fallible and evolving and so is our knowledge of how to acquire, process and apply knowledge. Boose (1989) has classified the current repertoire of knowledge acquisition tools, their limited strengths and major weaknesses. We do not claim that the knowledge support system described in this paper goes very far, or is significantly innovative, in overcoming those weaknesses. We do suggest that its heterogeneous architecture illustrates an enabling technology that is fundamental to overcoming those weaknesses in the time-scales, and over the breadth of applications, that are now required.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. We are grateful to many colleagues for discussions over the years that have influenced the research described in this paper, in particular John Boose, Jeff Bradshaw, and other colleagues at the Knowledge Acquisition Workshops world-wide.

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. (1986) **Expertise Transfer for Expert System Design**. New York: Elsevier.

Boose, J H. (1989) A survey of knowledge acquisition techniques and tools. **Knowledge Acquisition 1** (1), 39-58 (March).

Boose, J.H. & Gaines, B.R., Eds. (1988). **Knowledge Acquisition Tools for Expert Systems**. London: Academic 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.

Gaines, B.R. (1986). Cognitive models of human-computer interaction in distributed systems. **Proceedings of 1986 International Conference on Systems, Man and Cybernetics.** Vol. IEEE 86CH2364-8, pp. 1358-1362. Atlanta (October).

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). 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. (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). Organizational integration: modeling technology and management. Zunde, P. (Ed.) **Empirical Foundations of Information and Software Sciences II.** New York: Plenum Press.

Gaines, B.R. (1988c). Positive feedback processes underlying the formation of expertise. **IEEE Transactions on Systems, Man & Cybernetics**, SMC-18(6), 1016-1020 (November).

Gaines, B.R. (1989a) Social and cognitive processes in knowledge acquisition. **Knowledge Acquisition 1** (1), 39-58 (March).

Gaines, B.R. (1989b) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. **Proceedings of 6th International Workshop on Machine Learning**, pp.156-159. San Mateo, California: Morgan Kaufmann (June).

Gaines, B.R. & Boose, J.H., Eds. (1988). **Knowledge Acquisition for Knowledge-Based Systems**. London: Academic Press.

Gaines, B.R. & Shaw, M.L.G. (1980). New directions in the analysis and interactive elicitation of personal construct systems. **International Journal of Man-Machine Studies 13**(1) 81-116 (July).

Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. pp.47-66.

Goodman, D. (1987). **The Complete HyperCard Handbook.** New York: Bantam.

Musen, M. (1989). **Automated Generation of Model-Based Knowledge Acquisition Tools**. London: Pitman.

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).

Rappaport, A. & Gaines, B.R. (1988). Integration of acquisition and performance. **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.25-1-25-20, Banff (November).

Sharp, M. & Gaines, B.R. (1989). Knowledge acquisition for user support in heterogeneous networks. **CASE'89: Proceedings of the Third International Workshop on Coomputer-Aided Software Engineering.** pp.229-239. London (July).

Shaw, M.L.G. (1980). **On Becoming a Personal Scientist**. 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). Advances in interactive knowledge engineering. Bramer, M.A., Ed. **Research and Development in Expert Systems III.** pp.111-122. Cambridge: University Press.

Shaw, M.L.G. & Gaines, B.R. (1987) 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**. pp.30-1-30-19, Banff (November).

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

Steels, L. (1986). Second generation expert systems. Bramer, M.A. (Ed). **Research and Development in Expert Systems III**. pp.175-183. Cambridge: University Press.

Woodward, B. (1988) Knowledge engineering at the front-end: defining the domain. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop**. pp.37-1-37-25. Banff (November).

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

CLASSIKA: A Knowledge Acquisition Tool for Use by Experts

Ute Gappa

Karlsruhe University, Institute of Logic, P.O. Box 6980, D-7500 Karlsruhe, and
GSF Research Center, Medis-Institut, Ingolstädter Landstr. 1, D-8042 Munich-Neuherberg
West Germany
e-mail: Gappa@iravcl.ira.uka.de

One of the most promising strategies to overcome the bottleneck of knowledge acquisition is to provide powerful expert system environments which enable domain experts to formalize their knowledge without the help of knowledge engineers. Therefore an expert system tool must have a specific model of the problem to be solved. The knowledge acquisition system CLASSIKA and its problem-solving environment (shell) D3 are specialized on heuristic classification tasks. Their problem-solving model is based on a hypothesize-and-test strategy which integrates an explicite cost-benefit-analysis for data gathering, the evaluation of hypotheses by combining categorical, probabilistic and non-monotonic reasoning and temporal reasoning by interpreting follow-up sessions. The core of knowledge base development using CLASSIKA consists of first entering the symptom and diagnosis vocabulary into graphical hierarchies, second specifying local object information like question texts and answer alternatives by filling-in object-specific forms and third entering the relational knowledge between objects via configuring and filling-in different types of tables. So the expert need only know to use the problem-specific graphical forms.

1. Introduction

The quality of expert systems will increase if there are knowledge acquisition environments which enable experts to formalize their knowledge by their own. To reach this aim, a knowledge acquisition tool has to incorporate both a powerful model of the problem solving process in the domain or of a specific problem class as well as an easy-to-use interactive input, say graphical, "programming language".

Domain-specific knowledge acquisition tools have the best chances to be used by experts themselves as they have incorporated the terminology and way of thinking of experts in their domain. An example of such a tool is OPAL [Musen and al. 87] which mimics the protocols

experts use for treating cancer patients. While OPAL exploits a very specific domain model, knowledge acquisition systems for classification such as ETS [Boose 85], AQUINAS [Boose and Bradshaw 87], MORE [Kahn and al. 85], MOLE [Eshelman and al. 87], TDE [Kahn and al. 86] and our system CLASSIKA [Gappa 88] use a more general model of a problem solving type, for example classification. The advantage of the latter is that they can be applicated in many more domains than domain-specific tools.

Our research with the expert system shell D3 and its knowledge acquisition tool CLASSIKA results from experiences we made with our predecessor classification shell MED2 [Puppe 87a, 87b] (running in Golden Common Lisp on an IBM-AT), which was used to develop various large knowledge bases, e.g. diagnosing faults in the process of manifacturing computer-boards, the process of producing small rubber parts and painting cars as well as in the service and repair of defective gearing units, vaccuum pumps and motor car engines on an engine-test stand. The development and especially the maintenance of the knowledge bases, which was done by domain experts, were expensive for the companies as the developers had to learn the (awful) syntax of the internal knowledge representation and had too less help in surveying the knowledge base by a clearly arranged display. So we put a lot of effort into improving the comfortability and attractiveness of the knowledge-acquisition environment. The new tool CLASSIKA should need a minimum of time to get familiar with and should garantee the maintanance of knowledge bases during their life time as well.

CLASSIKA is not yet (August 89) fully implemented. The current basic version is just to be handed out. We have used it at the GSF Research Center to develop a knowledge-base for treating diabetes patients. We also have built up some small expert systems with domain experts in some few hours which indicates high acceptance by domain experts.

To introduce the classification model we use in our system, we first explain the key mechanisms and show a sample session of a final expert system in chapter 2. In chapter 3 we describe the main ideas of the knowledge acquisition tool CLASSIKA illustrating the process of building a knowledge base. We have a short discussion of CLASSIKA and a comparison to other knowledge acquisition systems in chapter 4.

2. The problem-solving shell D3

2.1. Overview

D3 is an expert system shell for classification tasks, of which currently only the component using heuristic knowledge (MED2) is completed. Its problem solving method is based on a hypothesize-and-test strategy: Starting with the initially given information or symptoms, the system generates hypotheses and pursues the best of them by asking for additional data. Questions are comprised to *questionsets* corresponding to leading symptoms, technical investigations or tests so that questions concerning the same subject are asked together like a small questionaire. So, on the basis of the current hypotheses, the most promising questionset is selected and asked to the user which is done by a cost-benefit-analyses on the level of the questionsets. In this way the knowledge of data gathering is largely separated from the reasoning knowledge of rating hypotheses and is not encoded in symptom-diagnosis-rules.

D3 also provides mechanisms for non-monotonic reasoning by being able to retract hypotheses and rules in the face of contradictionary evidence (e.g. exceptions from a rule) and for temporal reasoning by interpreting value-histories in follow-up sessions.
Figure 1 gives a look of D3's control strategy.

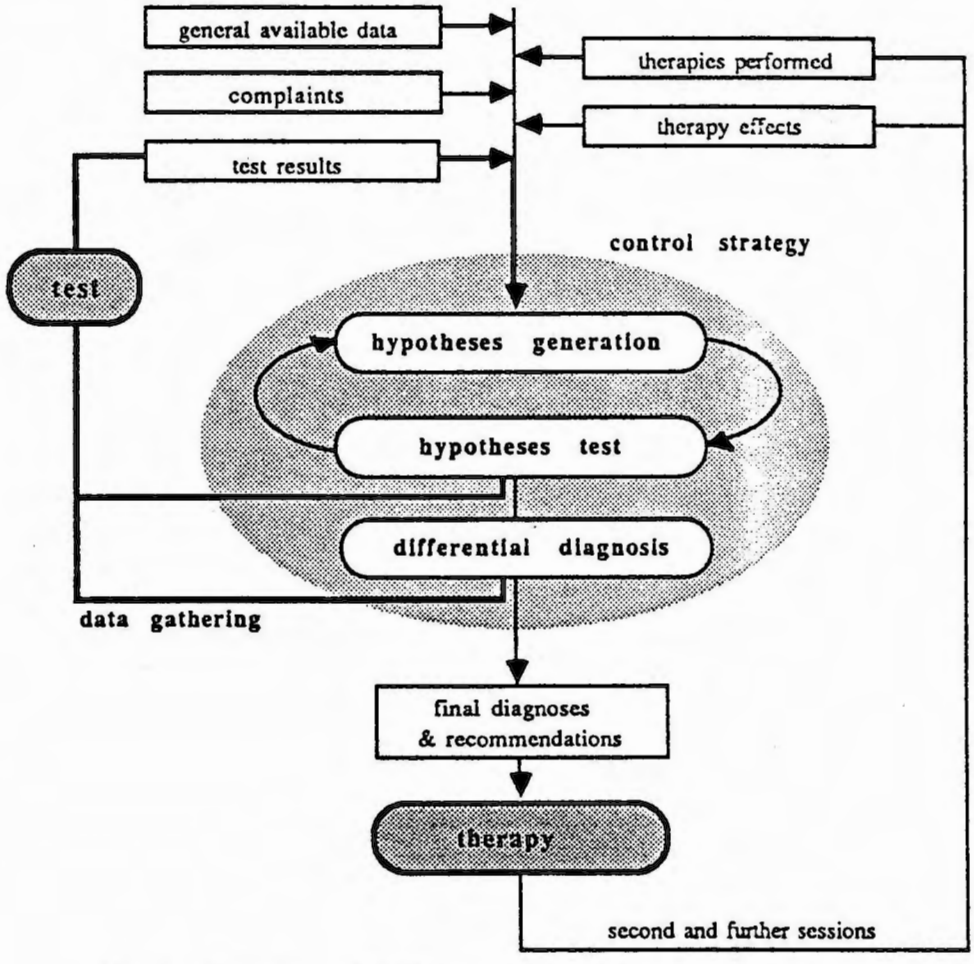


Fig 1: Overview of D3's problem solving mechanism

Hypotheses are evaluated by combining multiple criteria (compare figure 6) like categorical rules, probabilistic evidence, the predisposition of the diagnosis and its facility to explain given symptoms (explanation value). In more detail:

- *categorical information*: such as necessary, sufficient and exclusionary preconditions of the diagnosis
- *predisposition of the diagnosis*: The predisposition is derived from the a-priori probability of a diagnosis (its probability when not knowing any symptoms) which can be modified by rules (predisposition rules). For example, the a-priori probability that the spark plugs of a car are broken is comparatively high, and it will even be higher if it is known that more than 15,000 kilometers have been driven since their last change. The predisposition by itself cannot establish or exclude a diagnosis, it can only weaken or strengthen its probability.
- *probabilistic information*: A diagnosis can be rated using six positive and six negative probabilistic categories analogue to the ones in INTERNIST. Sometimes probabilistic reasoning can be replaced by specifying exceptions to a symptom-diagnosis-relation. For example,

instead of saying "symptom -> diagnosis in *70%*", it is more precise to say "symptom1 -> diagnosis *unless* symptom2".

- *differential diagnosis*: Diagnoses explaining a similar set of symptoms can be specified as differential diagnoses in the sense that usually only one of them will be true, e.g., in car engine either is »ignition timing too early« or »ignition timing too late«.
- *global point of view (plausibility check)*: Diagnoses that are able to explain observed, but still unexplained symptoms are more likely. Therefore their probability values are increased. For computing the explanation value of diagnoses, symptoms are grouped in characteristic units (called *explanationsets*) rated according to their importance, and associated to potential causes (diagnoses) which are able to explain them.

The system will terminate when either all symptoms are explained by established diagnoses or all relevant data for solving the case has been gathered. On the basis of the final diagnoses, therapy recommendations (represented as diagnostic variants) are selected either by default or by rules. If a performed therapy was not wholly successful, the user can start a new session entering his performed actions and the symptoms that have changed compared to the last session. Changed symptoms and their temporal evolution, influenced by therapeutic intervention, can then give rise to new hypotheses and improved management proposals.

In the following we illustrate a consultation session of a knowledge base. D3 is implemented in ALLEGRO COMMON LISP and runs on a Macintosh II. The expert system shown in the example gives advise if you have problems with your car and tells you which might be the fault [Walser 89].

2.1. A sample session of a consultation

At the beginning the user enters his major findings (complaints) by selecting appropriate terms out of a predefined hierarchy (hierarchy of questionsets). Double-clicking on a term results in the local expanding of the hierarchy (if the term is indicated by a point) or in selecting it (if the term is proceeded by an arrow).

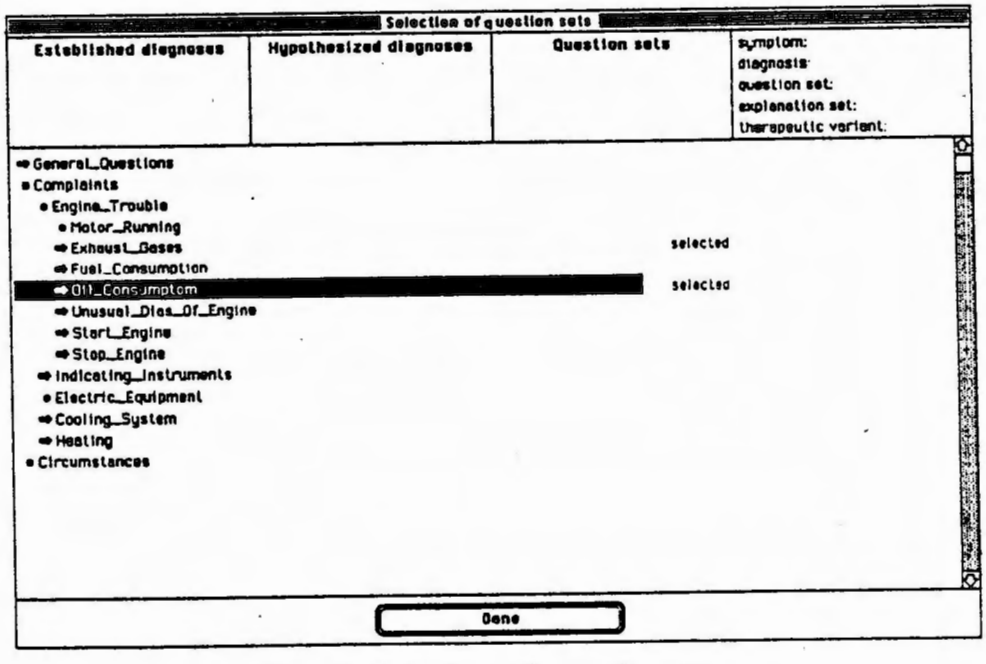


Fig. 2: Selection of questionsets

After the user's selection, the system displays the contents of the chosen questionsets. A questionset is similar to a questionaire and is answered mostly by clicking with the mouse onto the appropriate text.

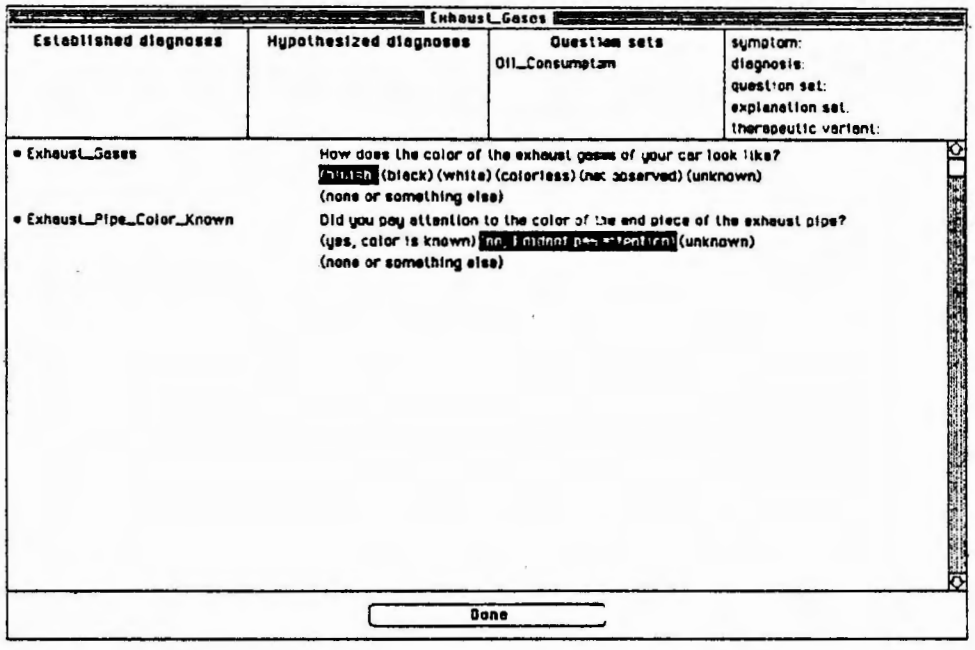


Fig. 3: Asking for the questionset "Exhaust_Gases"

The types of questions you see in the example are one choice and multiple choice questions, but there are also questions which ask for a number, a date (e.g., 2 weeks ago), a duration (e.g., for 2 weeks) or a frequency (e.g., 2 times a week).

The organization of the questions belonging to a questionset is also hierarchical. At the top of the hierarchy is the question which decides whether the rest of the questionset does make sense to ask for (see the question "Exhaust_Gases" in the example). If the given answer is not "none", "something else" or "unknown", the standard questions of the questionset are displayed underneath. In the example there is only one standard question. In dependence on previously given answers there there might also be follow-up questions.

Now, when the user finished "Exhaust_Gases", the system generates initial hypotheses, as there are "Cylinders_worn_out" and "Piston_Rings" in the example, while the next questionset chosen by the user, is displayed.

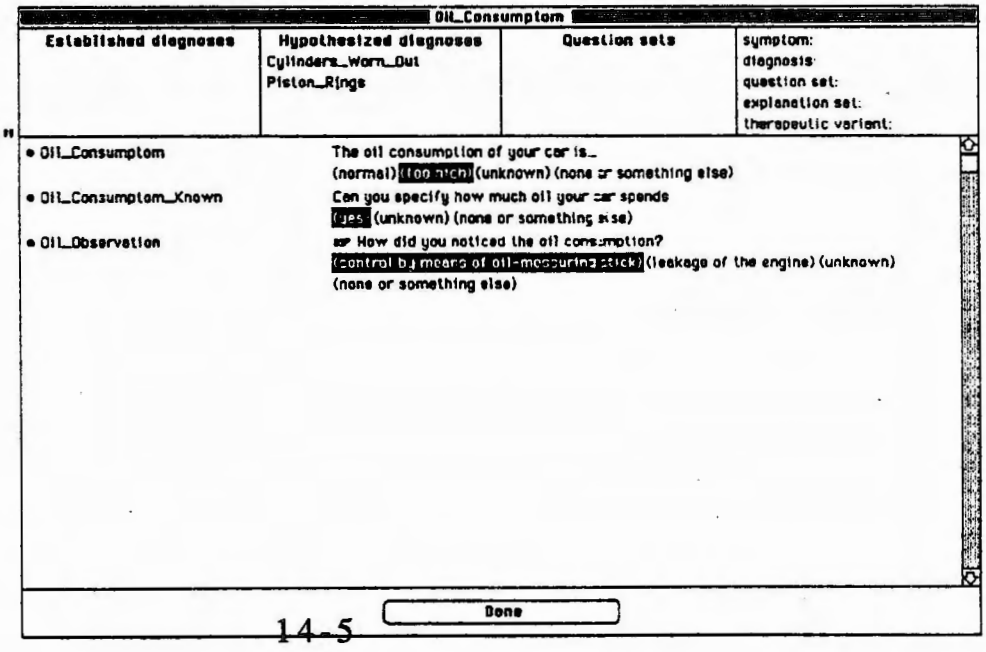


Fig. 4: Asking for the questionset "Oil_Consumption"

At this point the consultation stops and the system displays the final results ("Piston_Rings", see figure 6), as the knowledge base does not contain any technical investigations to further evaluate the current hypotheses.

At all points of the session the user can ask for an explanation of the system's behaviour, e.g. why a certain question or questionset is being asked for or why certain conclusions are drawn or also not drawn. He does it by choosing the appropriate command out of the menubar.

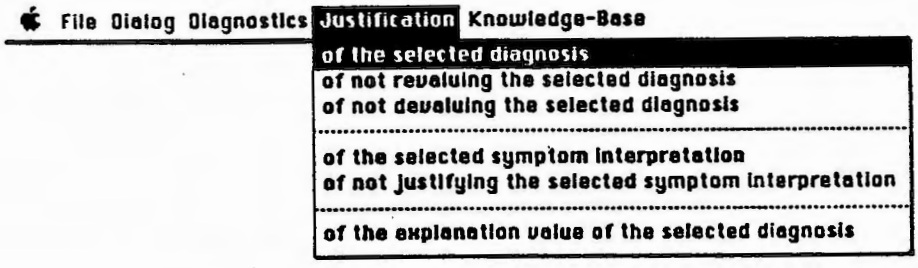


Fig. 5: Menubar with the explanation menu

We just show the explanation for the established diagnosis "Piston_rings", in which the various criteria for evaluating a diagnosis are indicated.

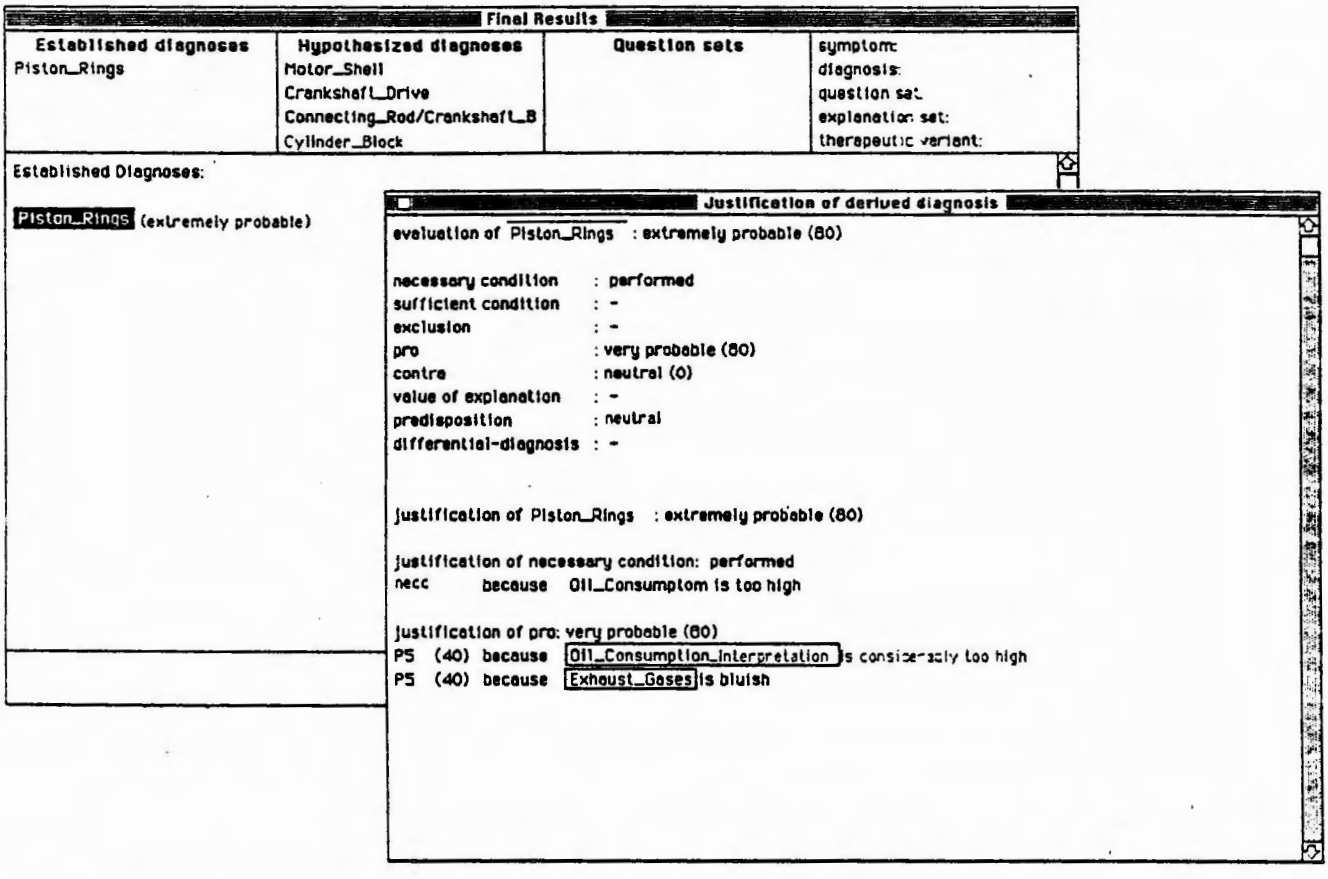


Fig. 6: Final results and justification of the established diagnosis "Piston_Rings"

3. Knowledge acquisition with CLASSIKA

The main aim of the knowledge acquisition tool CLASSIKA is it to enable domain experts to build knowledge bases of their own, without the assistence of an intermediate person. So one of the requirements for the development was that the user interface should especially adress computer layman and should therefore operate graphically and by clicking with the mouse. Another characteristic of experts is their lack of time, so the knowledge transfer must be as effective as possible. Therefore CLASSIKA provides a programming formalism on a high level of abstraction like special graphical hierarchies, object-specific forms and rule tables. These functional oriented input formats are abstractions of the object oriented frames and rules of the target knowledge representation and facilitate a better survey/inspection of the knowledge base. For example, if the expert wants to specify, how questionsets are indicated, he gets one table including all relevant information for this task; he does not have to manipulate all objects involved one by one, which was the case in our predecessor knowledge acquisition system.

3.1. Building the basic structure of a knowledge base

The minimal structure of a knowledge base for classification consists of findings (symptoms), diagnoses or classification results and of the diagnostic relationship between the two of them.
In CLASSIKA the vocabulary of the symptoms and diagnoses is entered graphically in hierarchies. The following figure shows the questionset hierarchy of the motor-car knowledge-base. Each object name of the hierarchy shown represents the start of a questionaire. The next layer of the hierachy would be the standard questions of the questionsets. The expert can enter these by selecting an object and choosing the command "Append questions..." out of the "Edit"-menu.

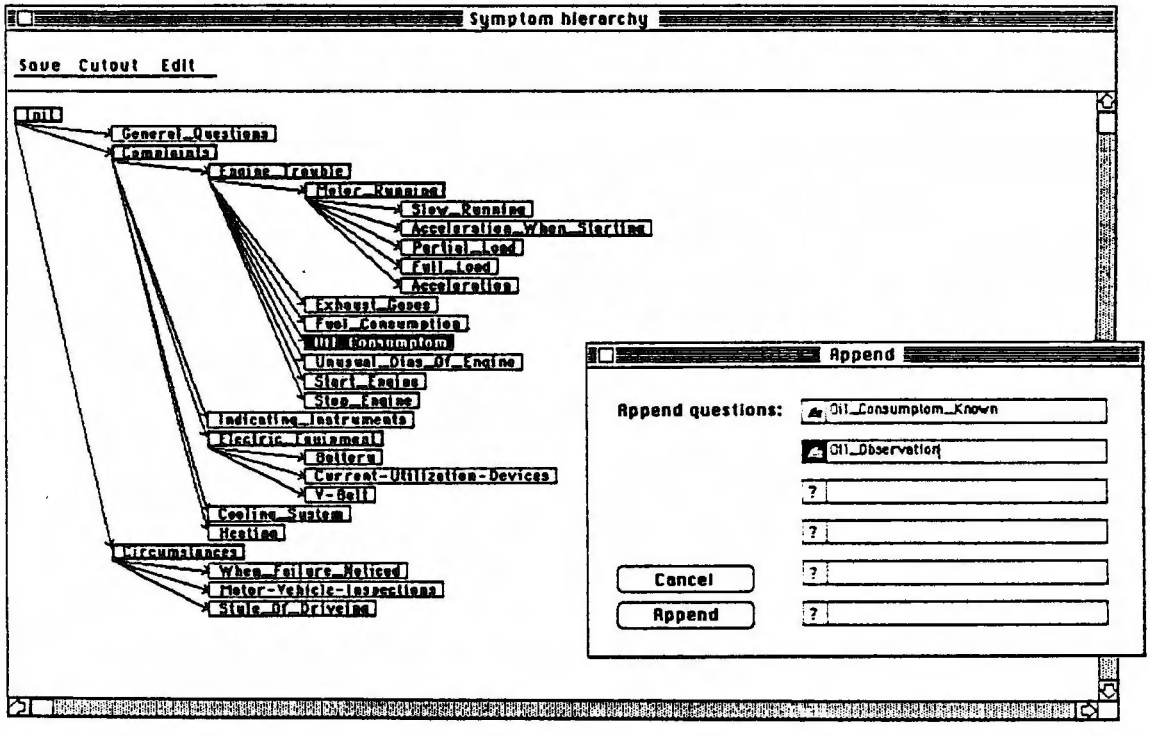


Fig. 7: Specifying the symptom hierarchy

If the expert wants to specify follow-up questions or local symptom interpretations, he includes them into the hierarchy and specifies the condition of asking resp. deriving them. For this purpose, a special rule-form is offered. You can see another rule form in figure 12; rule forms only differ in the properties *type* and *action.*

The expert enters the texts of questions and answer alternatives and additional information by clicking onto the object and choosing a command for getting the appropriate form out of the menubar. For example, if he clicks onto "Exhaust_Gases" the following form for questionsets will appear (see figure 8):

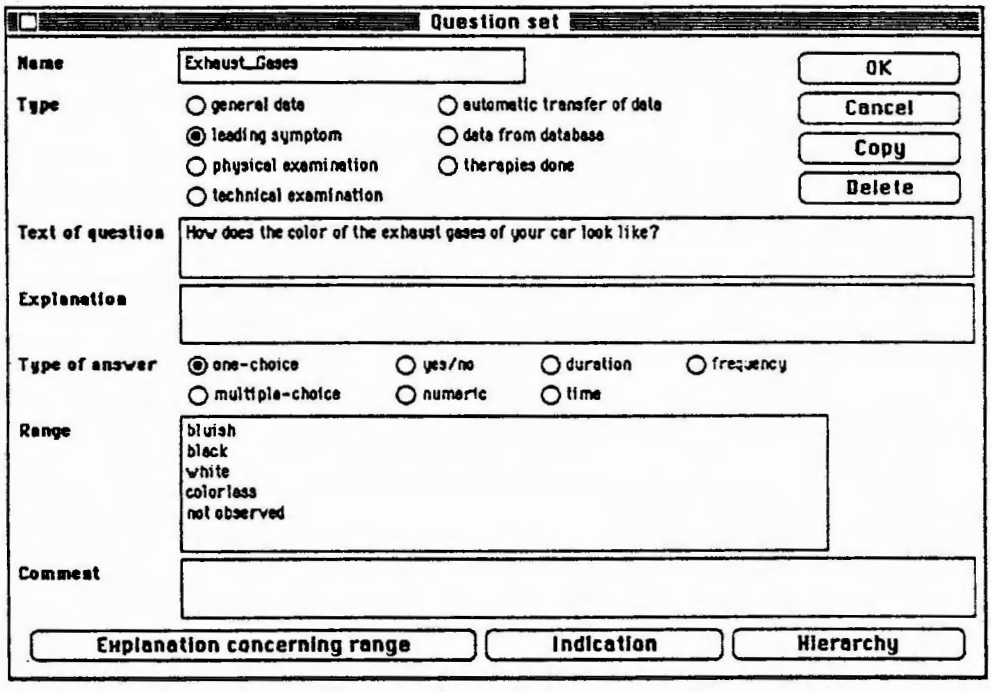


Fig. 8: Questionset form

In the form you have to select one of the possible types of a questionset and one of the possible answer types of a question. The buttons at the bottom of the form activate further specification forms, e.g., the button "Explanation concerning range" invokes a form to give explanations for the individual answer alternatives. During a consultation session, their existence is indicated by the symbol of a forefinger and can be requested by a click on the symbol (see figure 4). By means of the button "Indication" the expert is given a table, by which he can specify the questionset´s indication (figure 13).

Analogue to the symptom hierarchy the diagnosis hierarchy is being built. Figure 9 shows the beginning of the motor-car hierachy which contains about 100 possible faults.

After specifying the symptoms and diagnoses, the expert has to state how strong a given symptom argues in favor of or against a diagnosis. He therefore selects the diagnoses he wants to consider. They will appear in the rows of a symptom-diagnosis-table, while the symptoms appear in the columns. If rules have already been formulated, the expert might want to fill in the lines of the table with the already relevant symptoms by the command "Get lines" (figure 10).

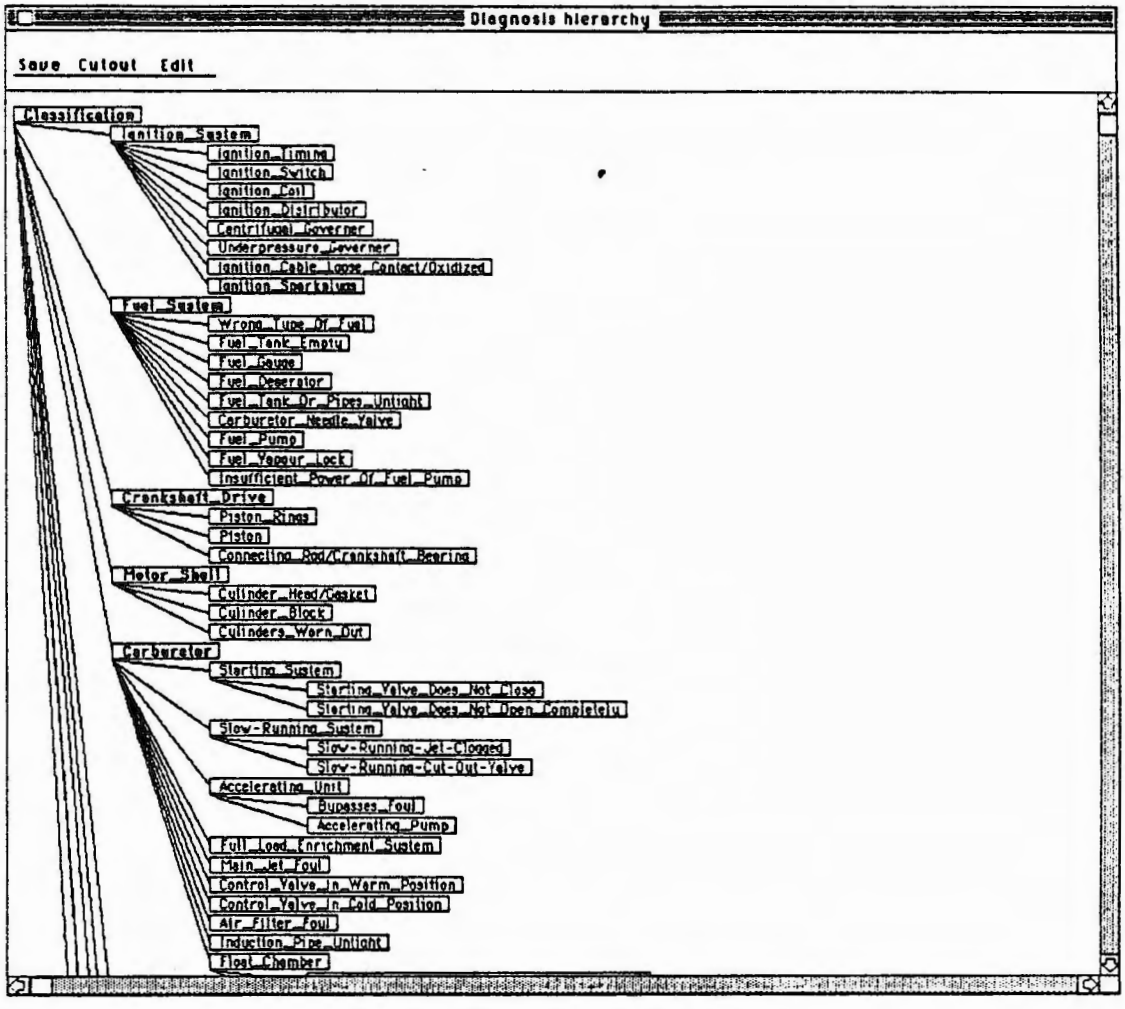


Fig. 9: Hierarchy of diagnoses

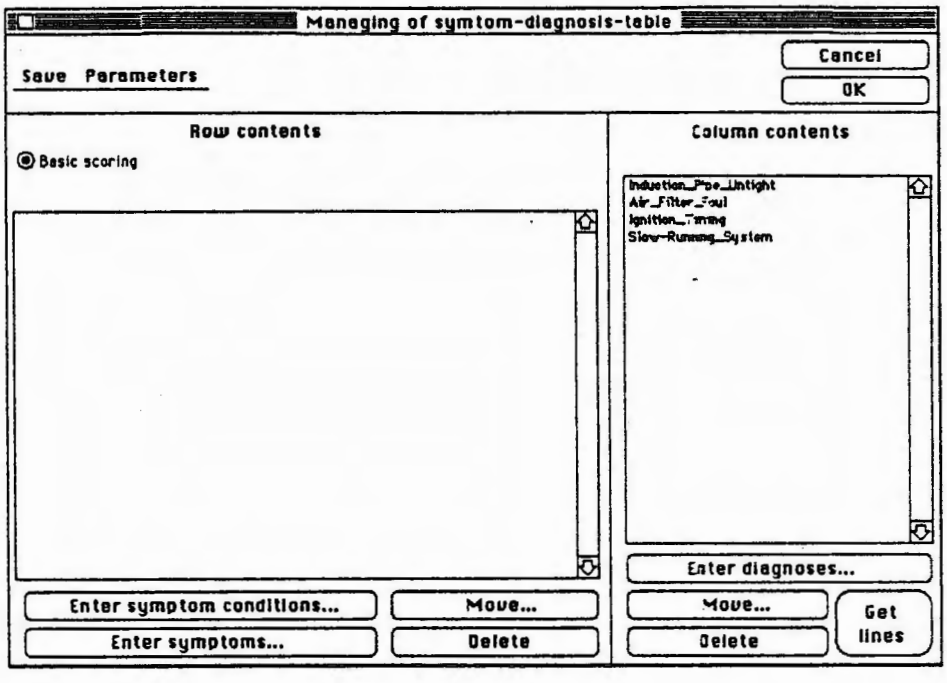


Fig. 10: Configuring a table

When the table is displayed, for every symptom-diagnosis relation two boxes appear. The first box refers to the existence of the symptom, the second to its non-existence, since the non-existence of a symptom can be a valuable information for excluding hypotheses and should be assessed. In one-choice symptoms ths is not necessarily needed as the non-existence is mostly itself named (as in e.g., low, normal, high).

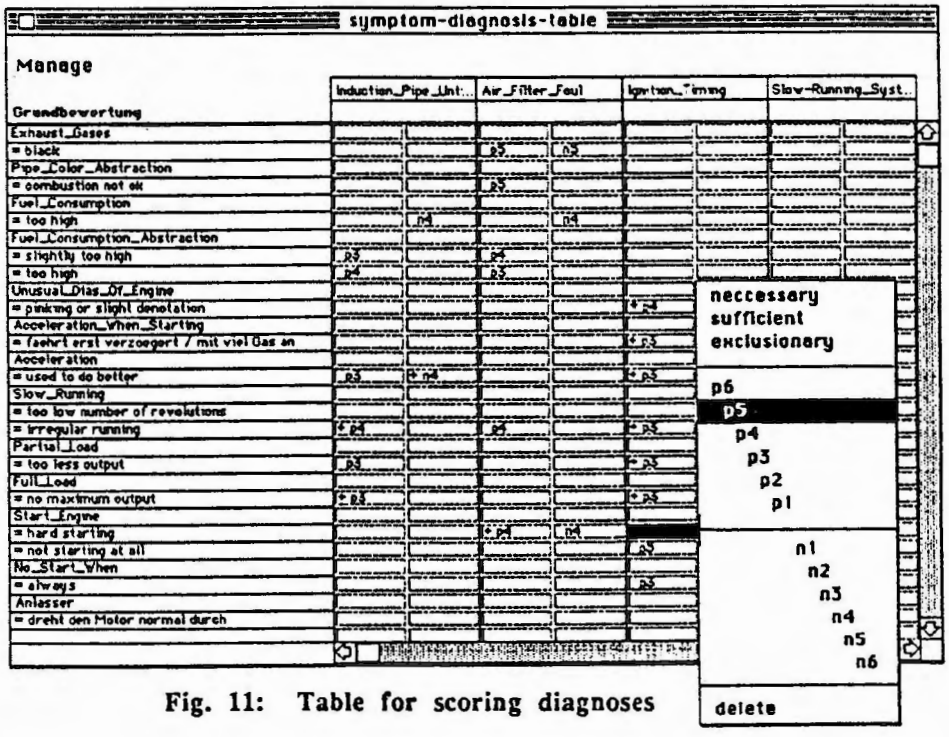


Fig. 11: Table for scoring diagnoses

In the figure above the rule "IF the *engine* is *hard starting*, THEN it is probable (by the rating *p5*) that the *ignition timing* is out of adjustment" is being entered via a pop-up-menu. Besides the categorical criteria *necessary*, *sufficient* and *exclusionary*, 6 positive and 6 negative point categories can be used in order to rate the diagnoses. Contrary to the repertory grids used in the systems ETS/AQUINAS and KITTEN/KSS0 [Shaw et Gaines 87] the expert directly specifies the symptom-diagnoses-evidence and not vice versa to make sure that he will be able to maintain the knowledge base later on. Every table entry is automatically converted into a production rule of MED2. As within the table only one symptom can be refered in the rule´s condition, a double-click onto the box effects the appearing of a rule form, in which multiple rule conditions can be specified as well as exceptions to the rule or activation contexts given (see figure 12). The existence of a more complex rule is indicated by a "+", the existence of multiple rules with the same first precondition by a "*" in the table boxes.

To specify many complex rules which combine multiple conditions, an alternative table can be used in which a rule is represented by a whole row rather than a box in the type of table shown above, and the symptoms can be marked and qualified.

The context of rules can also be specified globally for a whole table. In engine diagnosis for example, for each different engine type an own context can be defined, so that the special rules need only be activated in their context.

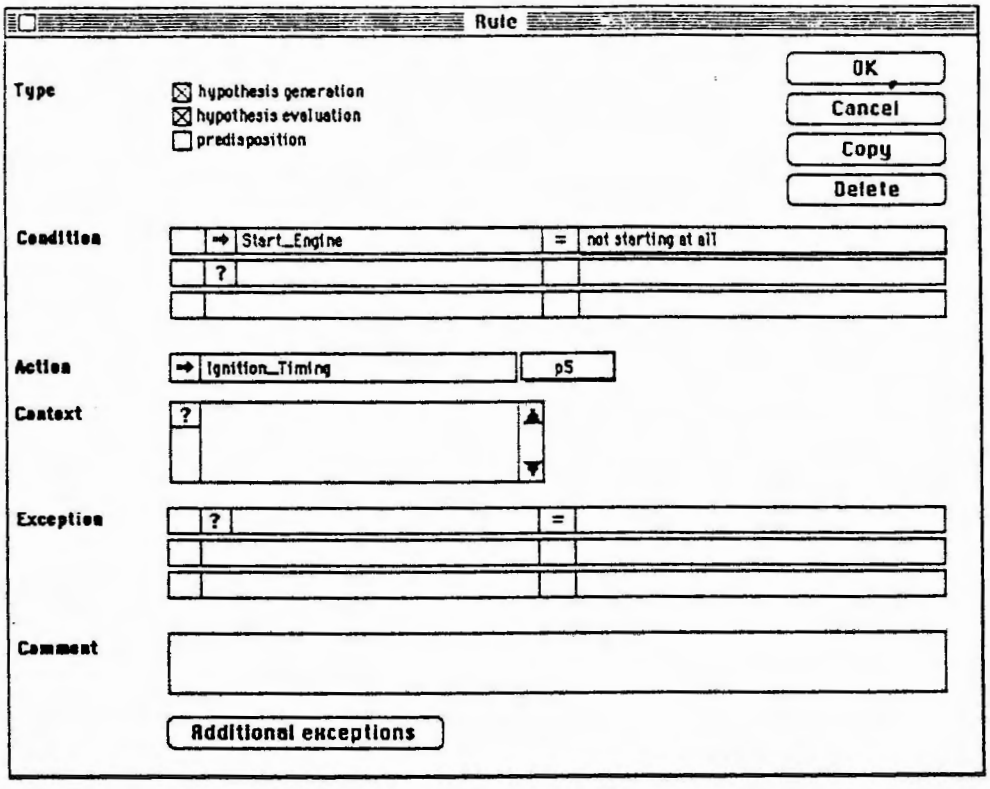


Fig. 12: Rule form

3.2. Extending the knowledge base by advanced knowledge types

The methods described so far are sufficient to acquire and represent just the basic structure of diagnostic problem solving. For modeling more advanced types of knowledge CLASSIKA offers additional types of tables (according to the different rule types in MED2), in which the expert can specify

- the control of data gathering (indication of questionaires) in dependence of the actual system state (e.g. current hypothesized diagnoses) (see fig. 13),
- computations of formulas from numerical values or automatic selections of answer alternatives, e.g. to abstract given quantitative symptoms to qualitative ones,
- plausibility checks of the input data for detecting inconsistencies,
- the selection of therapeutic variants (repair suggestions),
- the derivation of socalled *explanation sets* (comprising symptoms according to their possible explanation) and their correspondence to diagnoses which can explain them, so that the established diagnoses can be checked for plausibility.
- the correspondence of diagnoses and those explanationsets which can be explained by them (plausibility check of the established diagnoses).

Most of the above mentioned knowledge types contain uncertainties, which can be expressed by specifying the strength of the evidences in some way. For example, for the indication of questionsets a cost-benefit-analysis can be specified, which weighs the costs of gathering a questionset, e.g. performing a technical investigation, against the benefits for evaluating the current hypotheses. For doing this, the following table is provided:

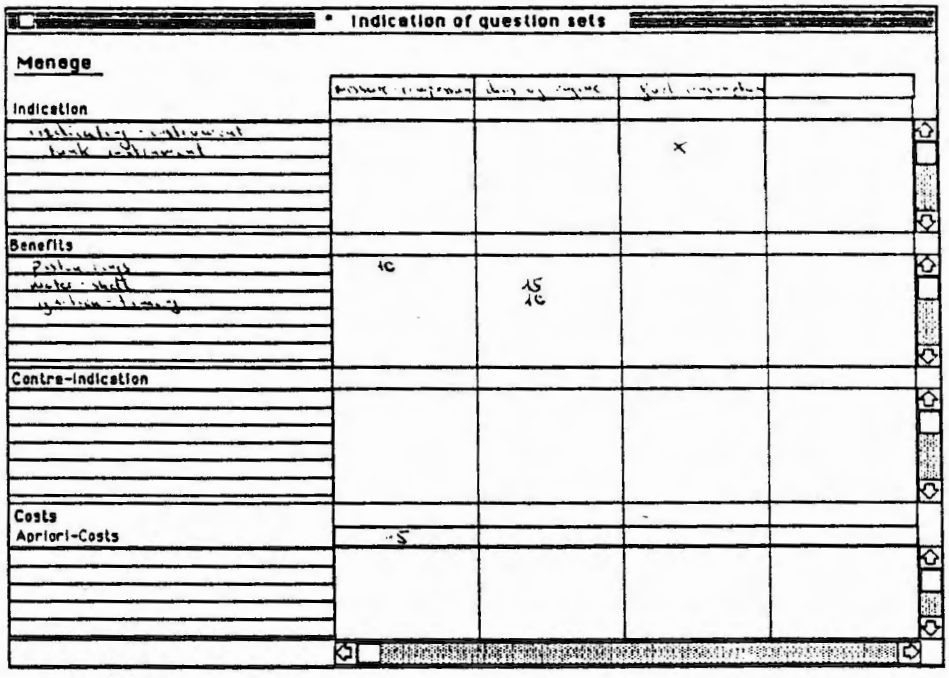


Fig. 13: Table for the indication of questionsets

To represent e.g. the risk of a patient to undergo a special test, the static costs of a questionset can be modified by rules. Questionsets can also be categorically indicated (e.g., "if a patient complaints about chest pain, then do an ECG") or contra-indicated (e.g., "if heart infection cannot be ruled out, fo not do an treadmill examination").

For selecting therapies, therapeutic variants can be instantiated which represent different therapeutic or repair modalities (figure 14).

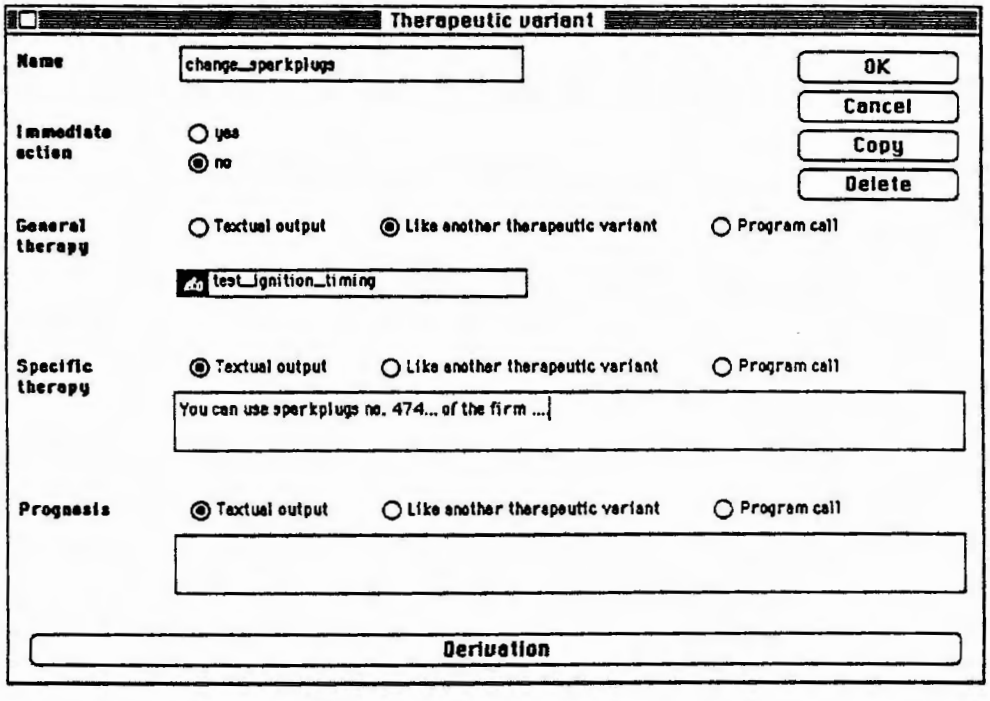


Fig. 14: Form for a therapeutic variant

To determine which therapy should be recommended, the expert may specify a questionset which will ask for special information, after the corresponding diagnosis is established, and use this data to rate the possible therapies. A therapeutic variant may also be selected by default, as we did in the following example.

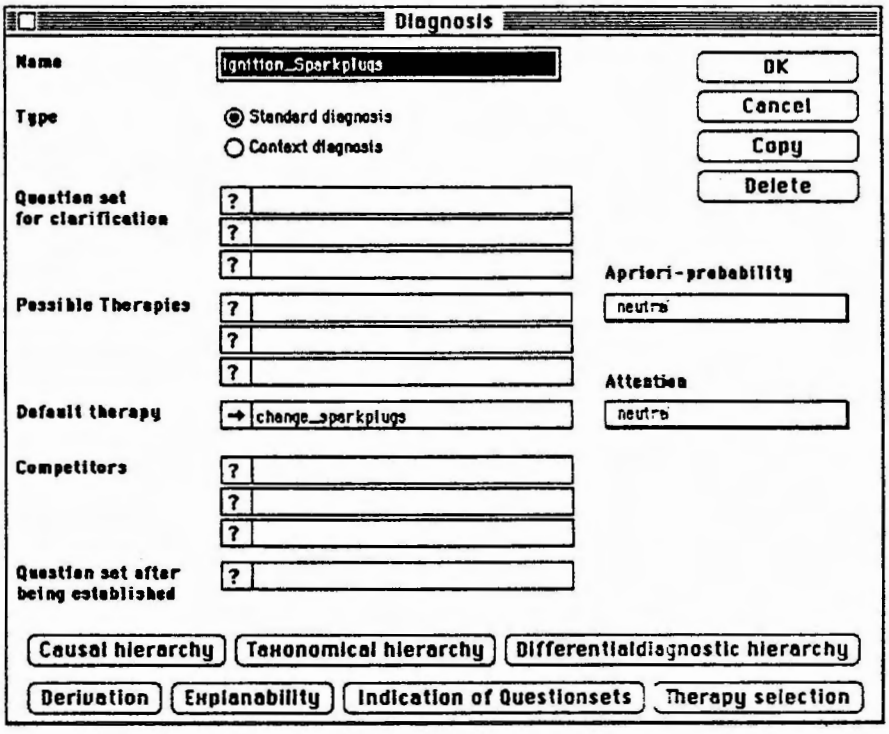


Fig. 15: Form for a diagnosis

If follow-up sessions should be supported, the expert has to define further questionsets which explore the details of therapies or repairs the user might perform. For the evaluation of value-histories of symptoms, the expert is offered special predicates for entering time-dependend conditions. With the help of them, he is supported to formulate rule conditions like "if the degree of fever is fallen by at least one degree within one day after taking a special medicine then...".

4. Discussion

The knowledge representation of an expert system plays a crucial role for knowledge acquisition. MED2´s highly structured representation provides a very strong and adequate model for meeting typical classification requirements. Knowledge concerning cost-effective data-gathering, generation of hypotheses and their evaluation by combining multiple criteria such as categorical conditions, probabilistic evidence, predisposition, differential diagnoses and explanation value, is explicitly represented. Additional mechanisms include checking the input data, proposing recommendations and evaluating follow-up sessions. Exploiting MED2´s model of classification, CLASSIKA can offer multiple knowledge acquisition techniques based on easy-to-understand input formats with a high level of abstraction.

CLASSIKA differs from a lot of other knowledge acquisition systems for classification in the richness of the kinds of knowledge which are acquired. Other systems often have a much

simpler knowledge representation covering only the core of diagnostic reasoning. A difference to TDE (resp. TEST [Kahn 87a]) is, as far as we can see, TDE does not support non-monotonic and probabilistic reasoning. We designed D3 especially for reasoning with uncertainty in respect to hypotheses evaluation as well as selection of diagnostic tests and therapies and handling exceptions. For example, in the domain of thoracic pain diagnosis [Puppe 84], with which our work began, only few symptoms are existing allowing categorical reasoning.

A well known knowledge acquisition technique is the repertory grid method as realized in KSSO (KITTEN [Shaw and Gaines 87] , RepGrid, NEXTRA), the Expertise Transfer System (ETS) and in AQUINAS. To a very large extent, these system facilitate experts by guiding the knowledge acquisition dialog and automatically generating rules out of the experts´ input. The model of the knowledge input is represented by a rating grid, e.g., a diagnosis-symptom table indicating the ratings of symptoms when a certain diagnosis is given. The diagnosis-symptom correlations are rather easy for experts to specify compared to the symptom-diagnosis correlations, e.g., the rating of how much the existence of a symptom indicates a special diagnosis. As the latter knowledge type (the symptom-diagnosis direction of the correlations) is needed for problem solving, they are derived from the table entries and by some additional data (e.g. the relative importance of individual symptoms). Although the automatic generation of rules is very convenient for the experts, it involves serious problems. Since the required assumptions for statistical computation of the rules and their probabilities are violated in most cases (e.g., that all diagnoses are included), the rules have to be checked and modified by the expert anyway. For this task, the expert is not supported and has to switch from ETS to a different tool, as sayed by the authors, when extending the knowledge base. This problem arises from the fact that the model of the experts´ input is different from the model of the target knowledge representation (the one of classification rules). Musen calls this situation "using an implicite model" [Musen 88]. To facilitate the maintenance of the knowledge base, we prefer an explicite model, where the expert directly determines the probability values of the classification rules. The model of the provided knowledge acquisition formats (e.g., tables) in CLASSIKA do not differ from the one of the target knowledge representation, they only differ in the level of abstraction. That means that they can be derived from one another without any inference and the expert has no problems in maintaining the target knowledge representation later on.

Another difference is that CLASSIKA gives the expert the initiative for knowledge base building while AQUINAS, MORE and its successor MOLE want the system to have the initiative. The latter use static and dynamic analysis methods for determining which knowledge to ask for. In a complex domain using a rich knowledge representation, it is not feasable to have the system ask for the information because there are too many possibilities for knowledge base extension as well as error correcting. Therefore, we want to support the expert in building the knowledge base by his own initiative. This requires that the theoretical model of the system is transparent to the expert and that the input formalisms are easy to use and on an adequate level of abstraction. Additionally, powerful explanation facilities are provided so that the expert can trace the problem solving process and locate incorrect or missing knowledge.

If the expert directs knowledge base building, knowledge transfer will be more effective, which is the main goal of knowledge acquisition tools.

Acknowledgements

Special thanks to Frank Puppe for his contribution to the CLASSIKA project and review of the paper. I also want to thank Annette Meinl, Karin Zipf and Andrea Bernhardt who implemented parts of the system and Peter Walser who developed the example knowledge base.

References

Boose, John H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology, International Journal of Man-Machine Studies, Vol. 23.

Boose, John H. and Bradshaw, Jeffrey M. (1987). Expertise transfer and complex domains: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems, International Journal of Man-Machine Studies, Vol. 26.

Eshelman, Larry L., and Ehret, Damien D., McDermott, John, Tan, Ming (1987). MOLE: a tenacious knowledge acquisition tool, International Journal of Man-Machine Studies, Vol. 26.

Gappa, Ute (1988). CLASSIKA: a knowledge acquisition tool facilitating the formalization of multiple aspacts in heuristic classification, European Knowledge Acquisition Workshop, GMD-Studien No. 143.

Kahn, Gary, Nowlan, Steve, and McDermott, John (1985). MORE: an intelligent knowledge acquisition tool, IJCAI.

Kahn, Gary S., Breaux Edwin H., DeKlerk, Peter and Joseph, Robert L. (1986). A mixed-initiative workbench for knowledge acquisition, Knowledge Acquisition for Knowledge Based Systems Workshop, Banff, Canada.

Kahn, Gary S. (1987a)., TEST: a model-driven application shell, AAAI.

Kahn, Gary S. (1987b)., From application shell to knowledge acquisition system, IJCAI.

Musen, Mark M., Fagan, Lawrence L. M., Combs, David M., and Shortliffe, Edward H. (1987). Use of a domain model to drive an interactive knowledge-editing tool, International Journal of Man-Machine Studies, Vol. 26.

Musen, Mark M. (1988). Conceptual concepts of interactive knowledge acquisition tools European Knowledge Acquisition Workshop, GMD-Studien No. 143.

Puppe, Frank (1987a). Diagnostic problem solving with expert systems (in German), Informatik Fachberichte 148, Springer.

Puppe, Frank (1987b). Requirements for a classification expert system shell and their realization in MED2, Applied Artificial Intelligence 1.

Puppe, Bernhard (1984). The history of computer-assisted medical decision making and MED1, an expert system for thoracic pain diagnosis (German), Dissertation, Universität Freiburg.

Shaw, Mildred L. G., and Gaines, Brian R. (1987). KITTEN: knowledge initiation and transfer tools for experts and novices, International Journal of Man-Machine Studies, Vol. 27.

Walser, Peter (1989). MODEX: An expert system for fault-finding in Otto carburetor engines (German), Practical study , Karlsruhe University.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Knowledge Acquisition From Natural Language For Expert Systems Based On Classification Problem-Solving Methods

Fernando Gomez
Department of Computer Science
University of Central Florida
Orlando, Fl 32816
gomez@ucf.csnet

Abstract

It is shown how certain kinds of domain independent expert systems based on classification problem-solving methods can be constructed directly from natural language descriptions by a human expert. The expert knowledge is not translated into production rules. Rather, it is mapped into conceptual structures which are integrated into long-term memory (LTM). The resulting system is one in which problem-solving, retrieval and memory organization are integrated processes. In other words, the same algorithm and knowledge representation structures are shared by these processes. As a result of this, the system can answer questions, solve problems or reorganize LTM.

1. Introduction

Can a simple expert system consisting of a hundred rules be designed from a natural language description by a human expert? Or can an expert system be updated from a natural language description of some of its components? In this paper, we present an integrated and domain independent system which builds classification-hierarchies from texts, retrieves information from them and solves problems.

We have been investigating the relation between comprehension and problem-solving. Our investigation has been guided by the idea that problem-solving and comprehension are not two separate processes each one with its own knowledge representation structures, but that they spring from a common source and share the same knowledge representation structures. It is beyond the scope of this paper to discuss these issues in detail, rather we center our discussion in providing evidence on how certain kinds of expert systems can be built from a natural language description by a human expert. We will show that the problem solving method used by the expert system to solve problems is the same as that used by the comprehender to keep memory organized and answer questions requiring chains of inferences.

We can roughly characterize the domain of the expert systems to which we have applied our method as consultation tasks. For instance, the tasks of building an expert system to select a programming language, or select a physician, etc. Some of these tasks are described in (Boose and Gaines, 1987). We can further characterize the scope of our system by the problem-solving method used to find solutions to a problem. The problem-solving method is that of classification problem-solving (Gomez and Chandrasekaran, 1984; Clancey, 1986). Classification-problem solving consists of conceptualizing the domain of the problem-solver into a hierarchy of concepts and designing an algorithm which solves a problem by placing it in the hierarchy of concepts. In (Gomez and Chandrasekaran, 1984) it is stated:

Given these principles of organization of medical knowledge, the solution of a medical

This research is supported by NASA-KSC Contract NAG-10-0058

case becomes a problem of taxonomic classification. It is similar to the problem of placing, say, a specimen of maple in a hierarchy of botanical concepts. It consists of identifying its superordinate and subordinate concepts.

The method to recognize problem solutions by placing them in a hierarchy of concepts was that of using clusters of production rules organized under the concepts in the hierarchy, called *specialists* (Gomez and Chandrasekaran, 1984). In this paper, (a) we will show how to automatically build the classification hierarchies from natural language, and (b) we will describe an algorithm which does not use production rules to place the problem solutions in the hierarchy.

Among the knowledge acquisition systems which have been developed, ETS (Boose, 1984) and AQUINAS (Boose and Bradshaw, 1987) are the closer to our approach in the sense that these systems are domain independent and are based on building classification hierarchies for the solution of the problems. However, the differences are remarkable since we are not using elicitation techniques to build the classification hierarchies, but we are building them from natural language descriptions. Also, we are not translating the expert's knowledge into production rules, but we are using an general algorithm to produce solutions from the constructed hierarchies. Nano-KLAUS (Haas and Hendrix, 1980) was an earlier natural language acquisition system. However, their system is not targeted to the construction of expert systems, and it does not deal with the issues which are essential to our approach. The differences from other knowledge acquisition systems, MORE (Kahn, Nowlan and McDermott, 1985), MOLE (Eshelman, Ehret, McDermott and Tan, 1987) and SALT (Marcus, 1987), are more striking since these systems do not use classification problem-solving and presuppose domain knowledge.

This paper is organized as follows. In part 1 consisting of sections 2 and 3, we briefly explain some of the ideas on which our method is based, and in part 2 consisting of sections 4, 5, 6, 7 we describe how these ideas are applied to the construction of expert systems.

2. Background

For the last few years, we have developed a model of comprehension of elementary scientific texts or expository texts and a program which embodies the model, called SNOWY, (Gomez, 1985; Gomez and Segami, 1989a, 1989b). The comprehender of our model is a reader who is unfamiliar with most of the concepts which he/she is reading about. When SNOWY starts reading a scientific paragraph its domain knowledge consists only of a bare hierarchy of concepts. As SNOWY reads, new concepts are integrated in the right place in the hierarchy and new relations about already known concepts are learned. For instance, after reading the passage below with no knowledge about hormones and insulin, SNOWY acquires the concepts *hormones, insulin,* and *level of sugar in the blood.*

Hormones are chemicals produced by animals. Hormones control the activities of cells. Insulin is a hormone produced by the pancreas. Insulin controls the level of sugar in the blood by making cells take in sugar.

SNOWY does not only acquire those concepts, but it also builds a classification hierarchy. For instance, the concept *insulin* is classified as a subconcept of *hormone produced by the pancreas;* this, as a subconcept of *hormones;* this, in turn, as a subconcept of *chemical*

produced by animals and, finally, this last one as a subconcept of *chemicals*. In contrast to knowledge-intensive models of comprehension which understand by framing the input in rich pre-built structures, SNOWY understands by using *formation* rules which build conceptual structures from the logical form of sentences. These conceptual structures are then passed to a *recognizer* algorithm which checks if those concepts exist in long-term memory (LTM). Finally, those concepts which the recognizer algorithm fails to recognize are passed to an integration algorithm which decides where to integrate those concepts in LTM. We have called these three phases concept formation, concept recognition and concept integration.

Although SNOWY is a knowledge-lenient model of comprehension, this does not mean that SNOWY does not possess any knowledge when it starts reading a text. Besides the linguistic knowledge which allows SNOWY to parse and disambiguate sentences (Gomez, 1988), SNOWY posseses a set of categories organized in a hierarchy, a set of primitive relations for action and descriptive verbs, which are used to build the logical form of the sentence, and a set of formation rules which build the LTM conceptual structures from the logical form of the sentences. Figure 1 contains a sample of the *a priori* hierarchy. The hierarchy is divided into physical things and ideas, as shown in Figure 1. The portion of the hierarchy which is shown in Figure 1 is strictly hierarchical. However, when the system begins to acquire new concepts, it builds a "tangled" hierarchy as described in the sections below.

As new sentences are typed, the system builds the representation of the new concepts and relations and inserts them into the hierarchy at the appropriate places. For example, for the sentence "germs are animates," the system adds the following to LTM:

(germ (is-a (animate)))
(animate (classes-of (germ)))

Before describing how to build these hierarchies, we need to explain briefly the knowledge representation structures used by our system. Three kinds of representation structures may be part of LTM: object-structures, action-structures, and event-structures. For purposes of this

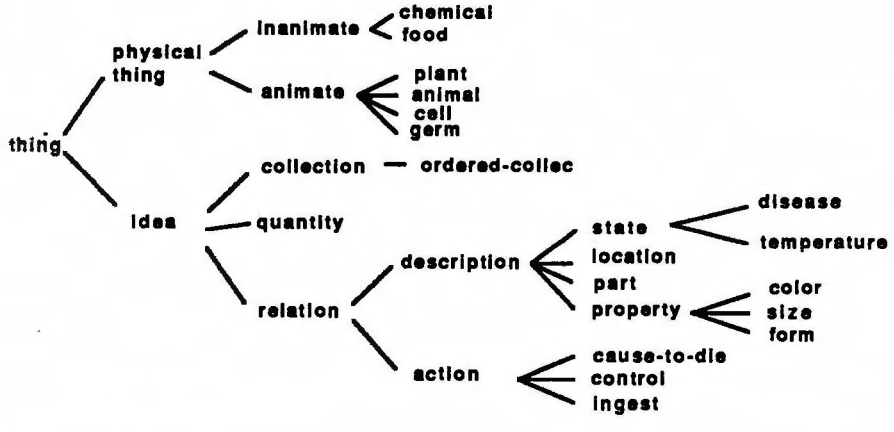


Figure 1: Partial content of the initial hierarchy of concepts in LTM

paper, we need only to describe object-structures and action-structures; event-structures are explained in (Gomez, 1986). All these structures are variants of the frame notation. An object-structure contains knowledge about physical or abstract objects and has slots called conceptual relations or predicates. Two kinds of conceptual relations can be distinguished: those which describe objects and those that attribute actions to physical things. This distinction corresponds to the distinction between descriptive and action verbs in natural language (Gomez, 1982). "Hormones are chemicals" and "all centipedes are slim and have many legs" are descriptive sentences, while "antibiotics kill germs" and "cells take in sugar" express actions performed by physical things. Descriptive attributes are also represented following the frame notation; for instance, the knowledge in the sentence about centipedes is represented as:

centipede
(form (slim (q1 (all))))
(has-body-part (leg (q1 (all)) (q2 (many))))

The slot q1 contains the quantifier of the concept described by the object-structure, and the slot q2 contains the quantifier of the second argument of the relation. Conceptual relations denoting actions are represented in the *object-structure* as:

relation (a1) (if the relation is monadic)

If the relation takes two arguments or more it is represented as:

(relation (concept1 a1))

where *concept1* is the second argument of the relation. If the same relation is true of two or more concepts, as is the case of the relation underlying the sentence "Penguins live in the sea and in the land," the relation is represented in the *object-structure* as:

(relation (concept1 a1) (concept2 a2) ... (concepti ai))

The terms *a1, a2, ... ai* stand for the names of the conceptual relations. The structure which represents the conceptual relation itself is called an *action-structure*. For example, the action-structure representing the conceptual relation in the sentence "antibiotics kill germs" looks like:

a1	**(inst a1 cause-to-die)**
(args (antibiotics) (germs))	**(actor a1 antibiotics)**
(pr (cause-to-die))	**(object a1 germs)**
(q1 (?))	**(quantifier-subject a1 unknown)**
(q2 (?))	**(quantifier-object a1 unknown)**

Of these two representations, we will use the one on the left side. The one on the right side is a slot-assertion notation (Charniak and McDermott, 86). The slot *args* contains the arguments of the relation. If the relation is monadic, the slot *args* will contain one concept. If the relation is diadic (as in this case), the slot *args* contains two concepts, and so on. The first concept in the slot *args* is always the actor of the action verb and the second concept is the object. The slot *pr* contains the relation. The slot *q1* contains the quantifier of the first argument, and the slot *q2* has the quantifier of the second argument of the relation. If the relation is triadic, there will be a *q3* slot for the quantifier of the third argument. The representation of "antibiotics kill germs" will be done by using the following object-structures and action-structure:

antibiotics **germs**
(cause-to-die (germs a1)) **(cause-to-die:by (antibiotics a1))**

a1
(args (antibiotics)(germs))
(pr (cause-to-die))
(how (?))
(q1 (?))
(q2 (?))

Every concept which fills a case (actor, object, destination, source, etc.) in the sentence is indexed independently. In the above example, both the actor, *antibiotics,* and the object, *germs,* are indexed using an object-structure. Note how the concepts *germs* and *antibiotics* point to the same a1-structure, so that the question "what kills germs"? can be answered. The question mark in the slots *q1 and q2* means that the sentence does not specify the quantifiers. If the sentence had said "all antibiotics kill germs," then the content of the slot *q1* would have been *all.* If the sentence had said "most antibiotics kill germs," the slot *q1* would contain *most.* Figure 2 contains the representation of "the human body kills germs by producing antibodies."

3. The Detection and Construction of Classification Hierarchies

The importance of explicit classification has been widely recognized and described by cognitive scientists working in reading. In explicit classification, the classes are clearly introduced and defined as in:

> **There are two kinds of mammals: terrestrial mammals and aquatic mammals. Aquatic mammals include the carnivorous and the omnivorous. The carnivorous includes the seals and walruses. The omnivorous includes the whales and etc.**

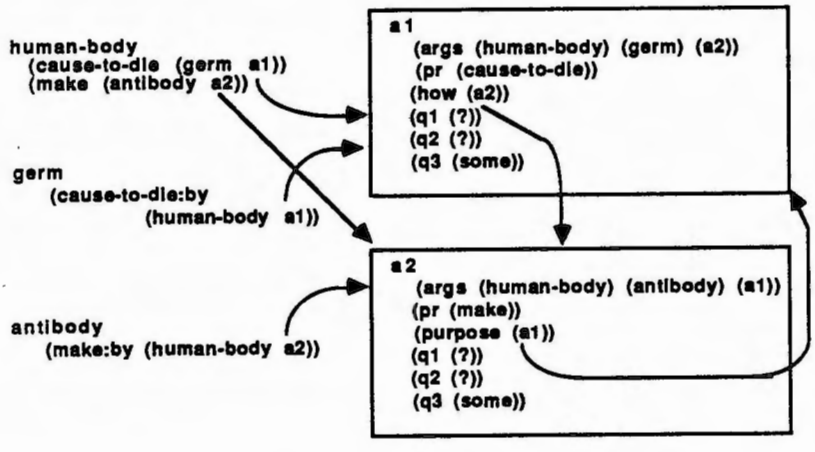


Figure 2: The representation of "the human body kills germs by producing antibodies"

In contrast to this type of classification, we will show that if a program is to acquire knowledge by reading texts it needs to classify in many instances in which classification is not apparent *prima facie*. Underlying classification is a more basic phenomenon than that which can be observed in other forms of classification, such as generalization-based classification or conceptual clustering. Furthermore, it is essential for constructing a problem solver from an English description of the task by a human expert, since the system will detect the classification hierarchies even when the expert does not see them. In the next sections, we will show that classification hierarchies underly two frequently used sentential structures: restrictive relative clauses and existentially quantified sentences.

3.1 Learning Concepts Denoted by Restrictive Relative Clauses

A first aspect of knowledge-acquisition is posed by the learning of concepts denoted by noun groups and restrictive relative clauses, e.g. "birds with long legs," "hormones produced by the pancreas," "programming languages which run in PCs," etc. The acquisition of these concepts involves their appropriate integration in the hierarchy, which in turn requires their classification by determining their parent and children nodes. For instance, consider the passage:

> **Birds with long legs live in swamps. They have a long bill to extract food from under water.**

This text introduces the concept *bird with long legs* for which a name is not given. However, a unique name in memory needs to be created for that concept so that additional knowledge about this concept can be learned by storing this knowledge under the same node, and so this node can be linked to the other nodes in the hierarchy. The acquisition of the concept *bird with long legs* entails its classification so that it can be integrated appropriately in the hierarchy of concepts. If the system fails to classify the concept, it will not be able to answer such questions as "are there birds with long bills?" or "do birds with long legs have feathers?"

Restrictive relative clauses and complex noun groups are represented in LTM via object-structures with a dummy name (a gensym), and their representation is characterized by the presence of a *characteristic-features* slot (written *cf*), whose purpose is to identify the concept by describing the features that are characteristic of the concept. For example, when building the representation of the sentence "insulin is a hormone produced by the pancreas," the concept *hormones produced by the pancreas*, a subconcept of the concept *hormones*, is created with the following representation:

> **(x1 (cf (is-a (hormone)) (make:by (pancreas (q2 (?))))))**
> **(hormones (classes-of (x1)))**

Within the *characteristic-features* slot there is always at least an is-a slot, which indicates the parent node of the concept being learned. In this example, the concept x1 has been created. X1 stands for a subconcept of the concept *hormones*, as indicated by the is-a slot, and is further characterized by referring to those hormones made by the pancreas. The acquisition of the concept x1 is completed by creating the slot classes-of in the concept *hormones*, and by filling it with x1. Had the slot classes-of existed in the concept *hormones*, then x1 would be added to it. The newly-created concept is identified by the content of the *characteristic-features* slot and not by the dummy name *x1*. Thus, if the question "what do hormones

produced by the pancreas cause?" is asked, the system recognizes that the concept *hormones produced by the pancreas* was previously created and accesses the object-structure x1. (The slot q2 contains "?" because it is unknown if every pancreas makes hormones.)

The acquisition of concepts described by complex noun groups is done in a similar manner. The representation of *bird with long legs* proceeds by creating the concept *long leg* first, and then the concept *bird with long legs*. Both concepts are represented in Figure 3. This definition creates the concept x1, a subconcept of the concept *leg*. The concept *bird with long legs* is represented by x2. The *characteristic-features* slot in x2 says that x2 is a subconcept of *bird* characterized by having long legs. Because the noun group does not specify how many long legs x1 has, the quantifier slot q2, whose scope is x1, has a question mark in it. If the system has knowledge about birds, it fills this slot; otherwise, it will remain with a question mark until the system finds out more about birds.

3.2 Formation or Learning Rules

The mechanism we have described for acquiring concepts denoted by restrictive relative clauses and noun groups is based on *formation or, learning rules*. These are very general rules anchored in the verbal primitives, and in some instances in lexical items. In most cases these rules use knowledge in LTM and in the structure built by the parser.

The formation rules which form the concepts x1 and x2, which have been explained above, are part of the formation of noun groups. For instance, the antecedent of the rule which is fired in the example above is:

> **The head noun has the feature animate and is followed by the preposition "with," and the object of the preposition has the feature body-part.**

Since the object of the preposition is also a noun group, the formation rules for the noun group are activated recursively to form the concept *long leg*. In this case, the antecedent of the rule activated is:

> **The head noun has the feature body-part and its modifier has the feature size.**

The latter rule will form the concept x1 and will pass it to the former rule, which will form the concept x2.

This has been a very simple example to illustrate basic formation rules. In the next section, we explain how formation or learning rules are used to learn more complex classification hierarchies.

x1	x2
(cf(is-a (leg))(size (long)))	(cf(is-a(bird)(body-part(x1 (q2(?)))))
leg	**bird**
classes-of(x1)	**classes-of(x2)**

Figure 3: The Representation of bird with long legs

3.3 The Acquisition of Classification Hierarchies

As we have seen, a classification hierarchy may be indicated explicitly, as in the sentence "antibiotics are chemicals" or it may be stated in a more subtle way as in the phrase "hormones produced by the pancreas." There is yet a more complex way of expressing a complex hierarchy of concepts like the illustrated in the passage below:

(1) Not all animals which live in the sea are fish. Some are mammals.

These two sentences introduce three concepts: *animals which live in the sea, sea fish* and *sea mammal.* The last two concepts are subconcepts of the first one.

The learning of these concepts and the hierarchy which they form is done by means of a learning rule attached to *not all* and by one of the learning rules anchored in *are.* The first sentence is parsed into:

(subj (not all animals) rela (g00001) verb (are) pred (fish))

g00001
(subj (animals) verb (live) prep (in the sea))

The learning rule which builds the representation for restrictive relative clauses is activated. This rule activates the formation rule for the verb *live.* The structure built is:

x1 **animal**
(cf(is-a(animal)(habitat (sea))) . **classes-of (x1)**

The name of the concept which has been created replaces the concept modified by the relative clause in the main sentence, resulting in:

(subj (not all x1) verb (are) pred (fish))

The formation of the main sentence proceeds by activating the learning rules for *are.* The following rule fires:

if the subject is under the scope of an existential
quantifier or the negation of an universal quantifier then

a. create a concept with a dummy name, say x2, and with
a characteristic-features slot containing the slots is-a (subj) and
is-a(pred)

b. create another concept with a dummy name, say x3,
and with a characteristic-features slot containing the slots is-a (subj)
and is-a (?).

The terms *pred* and *subj* in the rules are variables containing the subject and the predicate of the sentence, respectively. This rule when applied to (1) gives:

x2 **x3**
(cf (is-a (x1)(fish))) **(cf (is-a (x1) (?)))**

The application of the rule for restrictive relative clauses and the one for *are* has resulted in the creation of three concepts: the concept x1, which corresponds to *sea animal,* the concept x2, which corresponds to *sea fish,* and x3, which has been only partially specified.

Adverbs such as *almost, only,* and the the negation of universal quantifiers, e.g., *not all,* introduce links which connect the sentence where they appear to the next sentence. In order to solve these links, these lexical terms have attached to them formation rules. These rules are activated as part of the formation of the sentence which immediately follows the one which starts with *not all,* etc. These formation rules resolve the anaphoric reference of *some* in the second sentence and fill in the question mark in the is-a slot of the concept x3 above. The concepts and the hierarchy formed for the passage *Not all animals which live in the sea are fish. Some are mammals.* are found in Figure 4. The formation rules for *not all* do not only deal with this example, but also with examples such as:

Not all animals which live in the sea are fish. Whales are mammals.

In this case, the formation rule checks LTM to see if whales live in the sea. If so, it will replace the question mark in the concept x3 above with *mammal,* and create *whale* as a subconcept of x3. However, even if there is no knowledge about *whales* in LTM, the system infers from the structure of the sentence that *whales* are sea animals.

3.4 The Acquisition of Concepts Denoted by Existentially Quantified Sentences

The acquisition of concepts introduced by sentences starting with an existential quantifier requires the construction of a classification hierarchy in the same manner as we have done for relative clauses. The sentence "some mammals live in the sea" introduces the concept *mammal which live in the sea.* However, the representation in Figure 5 is inadequate for the job. The problem with this representation becomes obvious if the next sentence is "these mammals are intelligent." Since the structure above has not created the subconcept *mammal which live the sea* there is no node in the hierarchy to integrate the knowledge that they are intelligent. The representation of the sentence "some mammals live in the sea" is given below:

x1 (cf (is-a(mammal)) (habitat (sea))) --- is-a ---> mammal

X1 represents the concept *animals which live in the sea.* The concept *mammal* will contain

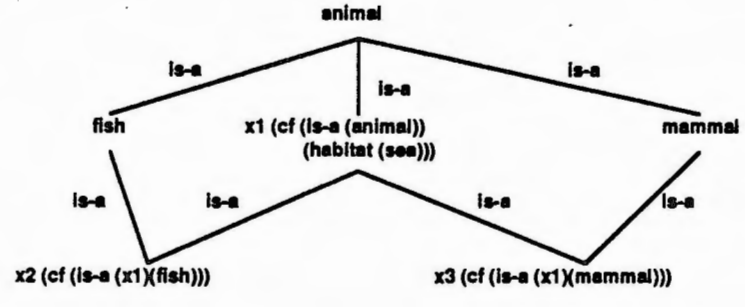


Figure 4: The hierarchy constructed for "Not all animals which live in the sea are fish. Some are mammals."

mammal **sea**
habitat (sea a1) **habitat: by (mammal a1)**

a1
args(mammal,sea)
relation (habitat)
q1 (some)
q2 (?)

Figure 5: Inadequate representation of "some mammals live in the sea."

the slot "classes-of (x1)." The knowledge expressed by the sentence "these mammals are intelligent" can now be integrated under the concept x1.

The role of classification is one of the most pervasive aspects in the structure of expository prose. As a final example consider the text below:

> Some birds are unusual because they cannot fly. The emu, the rea, and the ostrich are such birds. The emu is an Australian bird. The rea lives in South America. The ostrich lives in Africa and runs very fast.
>
> Some birds are unusual because they form a partnership with an animal. The oxpecker eats insects from the rhinoceros. The honey guide will lead a badger to a bees' hive.
>
> ---
> ---
> ---
> ---
> ---
> ---
> ---
> ---

The author found this text in a 6th grade comprehension exercise to teach children classification. The children were asked to write the names of birds on the dotted lines which follow the text. This text illustrates clearly that only through classification can the concepts and relations in the text be learned.

4. The Application of These Ideas to the Automatic Construction of Expert Systems

We will show in this section how these ideas can be applied to the construction of an expert system. The task which we will study is that of building a consultation system. We will use an example similar to that discussed in (Boose and Bradshaw, 1987), the selection of a programming language.

The expert is instructed to present his/her subject in a top down fashion describing first the most general concepts and then refining them until reaching the most concrete concepts which will form the tip nodes in the hierarchy. The paragraph below is just one way of how this

description may be accomplished. Many other are possible. After reading this paragraph, SNOWY builds the hierarchy of concepts shown in Figure 6.

> **There are several kinds of programming languages. Programming languages which manipulate symbols, called symbolic languages, are used for AI applications, while numeric languages are used for scientific applications. There are two main symbolic languages: Lisp and Prolog. Lisp is used for general AI applications, whereas Prolog is used when backtracking is needed. Common Lisp, Franz, Scheme and Golden Hill are Lisp languages. Franz runs in minicomputers and takes more than 3 megs of memory. Scheme is a good dialect of Lisp for PCs. It takes 0.8 meg of memory. Golden-Hill is also a version of Lisp for microcomputers, but it requires 2 megs of memory. There are several versions of Prolog, such as Quintus, Turbo and Mprolog. Of these, Turbo runs on PC's, and requires only 0.5 megs of memory. Numeric languages . . . .**

The system proceeds by parsing the sentences. In the current implementation, the system can parse any sentence containing unknown nouns and adjectives, but verbs must be known to the the parser for a succesful parse to occur. If the parser fails then the system asks the user for a definition of those words which are unknown to the system. First of all, the system asks the user if the word in question is a verb (the user can check that by consulting a dictionary). If the word is a verb the system obtains the syntactical usage of the verb by activating an interface which allows users without a formal knowledge of English grammar to add new verbs to the parser. The system can make sense of many sentences containing words unknown to it. We postpone the discussion of how this is done until the next section. We briefly explain some of the formation rules used to construct the hierarchy in Figure 6. The first sentence does not create any structure. From the second sentence, the system creates the concepts *symbolic language* and *numeric language*, according to the techniques which we have explained for the formation of concepts denoted by restrictive clauses and noun groups. The *characteristic-features* slot of the concept *symbolic language* contains the conceptual relation "manipulate symbols." Note that from this moment on, the meaning of *symbolic language* is a

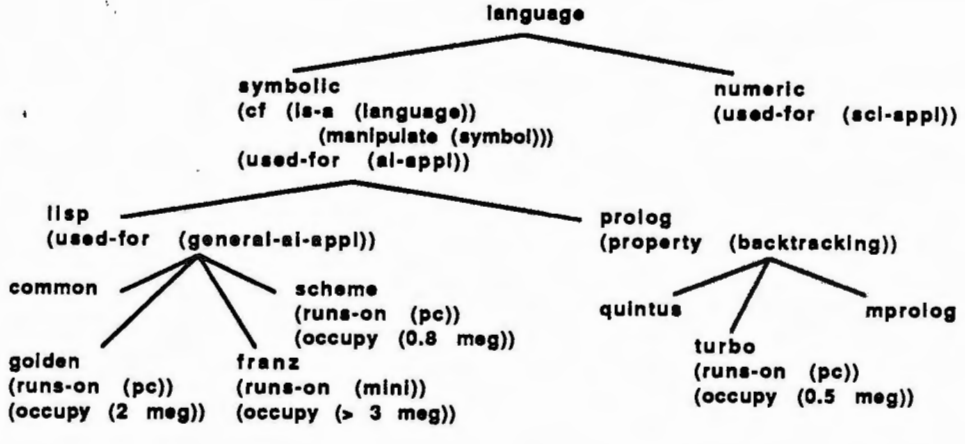


Figure 6: Hierarchy for the Text on Programming Languages

programming language which manipulates symbols. The formation of "symbolic languages are used for AI applications," is integrated by creating the descriptive relation "used-for (ai-application)" under the concept *symbolic language*. The next sentence creates two subconcepts of *symbolic language:* Lisp and Prolog. The next sentence differentiates the two languages. The formation rule which builds the LTM representation of "Prolog is used when backtracking is needed" constructs the following structure:

Prolog
(property(backtracking))

This a general rule which forms all logical forms which fall under the schema:

<somebody> <use> <obj1> when/if/in case that <somebody> <need> <obj2>

into the structure:

obj1(property(obj2))

The rule is using the inference that if something, say *x*, is used when something else, say y, is needed then *y* is a property of *x*. This is necessary if requests such as "I need a symbolic language with backtracking" are to be answered by the consultation system.

5. Defining Unknown Words

One of the critical issues in the design of the system is how many concepts in the user's input need to be previously known by it in order for the system to understand the text and solve problems. If the knowledge acquisition system is going to be domain independent, then the system can not have *a priori* knowledge about concepts which are domain specific. For instance, the concept of "backtracking" in the sample text can not be part of the *a priori* concepts. However, if the knowledge acquisition system is going to deal with a concrete domain, then one can provide it with a rich set of domain dependent concepts. Since our goal is to build a domain independent knowledge acquisition system, the problem of unknown concepts becomes critical. The sample text can be processed by the present implementation without knowing anything about the following concepts: "programming languages," "symbols," "AI," "numeric languages," "backtracking," "Lisp languages," "meg" and "Lisp," Prolog," etc. If one asks the system "what is backtracking," this will reply by saying "I do not know what it is, but I know that it is a property of Prolog." The system's reply to a question is "I do not know" if the concept in the question is not linked to the concepts in the *a priori* hierarchy. For classification problem solving, the system does not need to have a deep understanding of what backtracking is. In any case, the decision of determining if a concept should be fully known to the system rests on the user. We will describe below some ways in which the user can achieve this.

Let us go back to the acquisition of the other concepts in the text. The mechanism for acquiring the concept "meg" differs from the acquisition of the other concepts in the paragraphs. The sentence in which the concept "meg" appears for the first time is "Franz ... takes more than 3 megs of memory." Let us assume that the system knows that memory is something where information is stored. The problem here is that when the system is going to form the final knowledge representation from the logical form of the sentence, it can not determine the

meaning of "takes" because it does not know the meaning of "meg." One of the meanings of "takes" is to occupy space or storage. A rule attached to "take" says: "if the object of "take" is unknown, but is numerically quantified then assume that the meaning of "take" is to occupy space." This rule will infer that meg is a unit of measurement and will ask the user to confirm its inference.

If a user deems that the system should have a concept classified in the hierarchy, or the system fails to build the representation of a sentence because some key concepts are missing, then the user can define these concepts by using the methods explained below.

Synonyms: This is the most linguistic and superficial of the methods. This method does not place concepts in the hierarchy. It is also the easiest to use, although, of course, it has limitations. For instance, the word "dialect of" in the middle of the paragraph may be unknown to the system. An easier way to define this word is to say that it is a synonym of "kind of," which the system clearly knows because it appears in the first sentence of the paragraph. Note that the words need not to be synonyms in all contexts, but just in the one the user is dealing with.

Classification: The best way to define a word is by providing the upper-concept of the concept denoted by the word to be defined. For instance, the word *AI* can be defined by saying "AI is a computer science." An inappropriate definition will be to say something like "AI tries to make computers intelligent," since this does not place *AI* in the hierarchy of concepts which forms the basis of SNOWY's understanding process. Likewise, the definition the system is expecting of *backtracking* is not a long and intricate one, which probably will be irrelevant to the expert system being built, but just something like "backtracking is a programming technique" which places *backtracking* in the hierarchy of concepts. We are extending the *a priori* hierarchy of categories to include a large set of concepts so that it will be easy to place new concepts in the hierarchy by classifying them.

Part-of Definitions: Some words may be defined by using a *part-of* relation. For instance, the word *cpu* or *computer memory* can be defined by saying that they are part of a computer.

Functional Definitions: We are studying short functional definitions of concepts from which the system can infer the *part-of* relation plus some other information. For instance, the concept *computer memory* above can be defined as: "memory is something to store information into a computer." From this, the system can infer that *computer memory* is a physical thing, part of a computer and is used to store information. The reason why this method of definition has not been incorporated in the system, yet, is because it opens the door for a user to give a definition too complex to be understood.

6. Problem-Solving

After SNOWY has read the paragraph of section 4, a user interested in a consultation about programming languages may, at this point, run a consultation test. For example, the user may say "I am interested in a language that runs on PC's and is good for ai applications." Given this inquiry, the task of the system is to find the languages that have the features "runs on PC's" and "used-for ai applications." The class of languages that have these features is represented as:

x1 (cf (is-a (language)) (runs-on (pc)) (used-for (ai-appl)))

If this concept were present in LTM, the question would be answered by simply retrieving its children. Since this is not the case, one way to find the answer to the question is to *classify* the concept *x1* ("languages that run on PC's and are good for ai") within the current hierarchy of concepts. This task is done by a *classifier algorithm*, which determines where in the current hierarchy a new concept, such as *x1*, would be located if it existed in LTM, that is, the *classifier* determines which of the concepts currently in LTM are the parents of *x1* and which ones are its children. We briefly illustrate the steps taken by the *classifier*, using the concept *x1* as an example. (See (Gomez and Segami, 1989a) for a detailed description of the algorithms in this section and a comparison to other classifier algorithms). Given the representation structure of *x1*, the concept within its *is-a* slot determines what part of LTM needs to be considered by the *classifier*. In this case, only the concept *language* or its children can be the parents or the children of *x1*. The *classifier* performs a traversal of selected nodes under the concept *language*. The concept in each visited node is compared with the concept *x1*. Depending on whether one is a subclass of the other, or there is no hierarchical relationship between them, the *classifier* decides which node to visit next. The actual comparison of concepts is performed by the *compare-classes* algorithm, which is based on the following idea. A concept *x2* is a subclass of a concept *x1*, if each characteristic-feature of *x1* is true of each entity in the class *x2*. In our example, concept *x1* is first compared with the concept *symbolic languages*. No hierarchical relationship is found between these two concepts. However, one of the *characteristic-features* of *x1*, "used-for ai applications," is found to be true of all the descendants of the concept *symbolic languages*, so that, when the nodes under *symbolic languages* are examined, only one more feature needs to be verified: "runs on PC's." After doing this, it is found that *scheme, golden,* and *Turbo* run on PC's. Then, the *classifier* returns the list "((language) (scheme golden turbo))" to indicate that the concept "languages that run on PC's and are good for ai" is a child of the concept *language* and that it has the children *scheme, golden,* and *Turbo*. The system, then, would reply to the user as follows: "the following languages run on PC's and are good for ai: scheme, golden, and turbo." If the user decides to narrow down his/her choices, he/she may either specify additional features, or formulate questions. For example, the following interaction may now take place:

user:
  the language must run in a computer with less than 1 meg of memory
system:
  scheme and turbo require less than 1 meg of memory
user:
  what are the differences between scheme and turbo?
system:
  scheme is a lisp dialect used for general ai applications, while turbo is a prolog dialect used for backtracking

The first sentence typed by the user simply adds one more requirement to the desired programming language. Since we have an initial set of solutions, the new requirement needs to be verified for the members of this set only. This reduces the possible solutions to two. The second sentence typed by the user is a question, which requires that a comparison be made

between the features of *Scheme* and *Turbo*. This is accomplished by traversing the hierarchy upwards, from the nodes corresponding to *Scheme* and *Turbo*. The traversal stops when the two paths cross each other.

The *classifier* is the component that allows SNOWY to maintain a correct organization of its memory. Each new statement entered by the expert may cause the insertion of a new concept in the hierarchy, or it may trigger a reorganization of the concepts in LTM. Let us suppose, for example, that the expert continues entering information to the system, by typing the following:

Some languages are procedural. Procedural symbolic languages are widely available.

The sentence "some languages are procedural" is integrated as explained in section 3.4. SNOWY's long-term memory configuration for the two sentences is depicted in Figure 7.

If the expert now types the statement: "Lisp is a procedural language," the concepts in LTM will be reorganized as shown in Figure 8.

Thus, in an inquiry of the form "a language that runs on PC's, is good for ai applications, and is widely available," only the Lisp dialects will be considered to be possible solutions.

As one can see the problem-solving method which we have illustrated uses only categorical knowledge (Szolovits and Pauker, 1978) and not probabilistic knowledge. It is our belief that probabilistic reasoning plays a minor role in most problem-solving situations if memory organization and indexing form part of the problem-solving method. In any case, certain aspects of probabilistic reasoning need to be included in our system. Experts use sentences

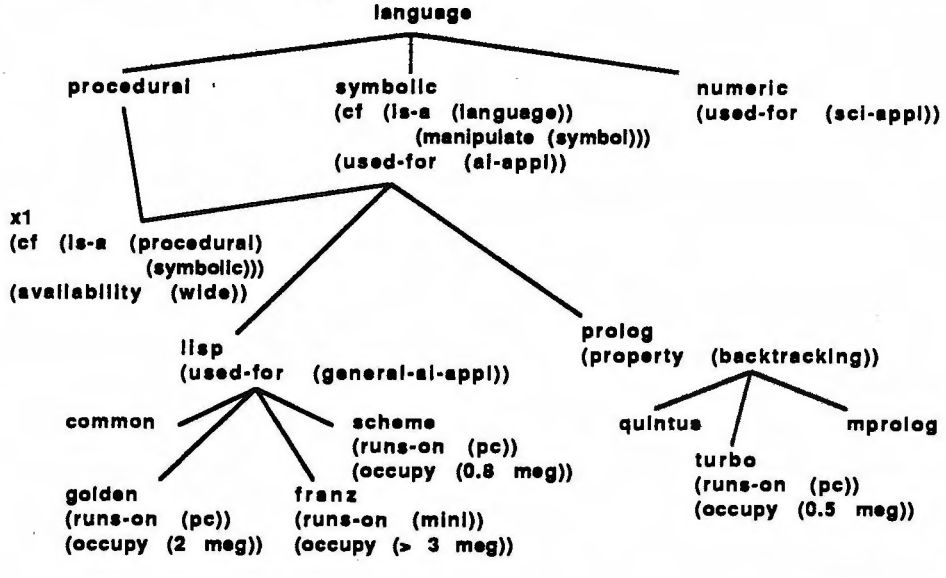


Figure 7: Organization of LTM after processing the passage "Some languages are procedural. Procedural symbolic languages are widely available."

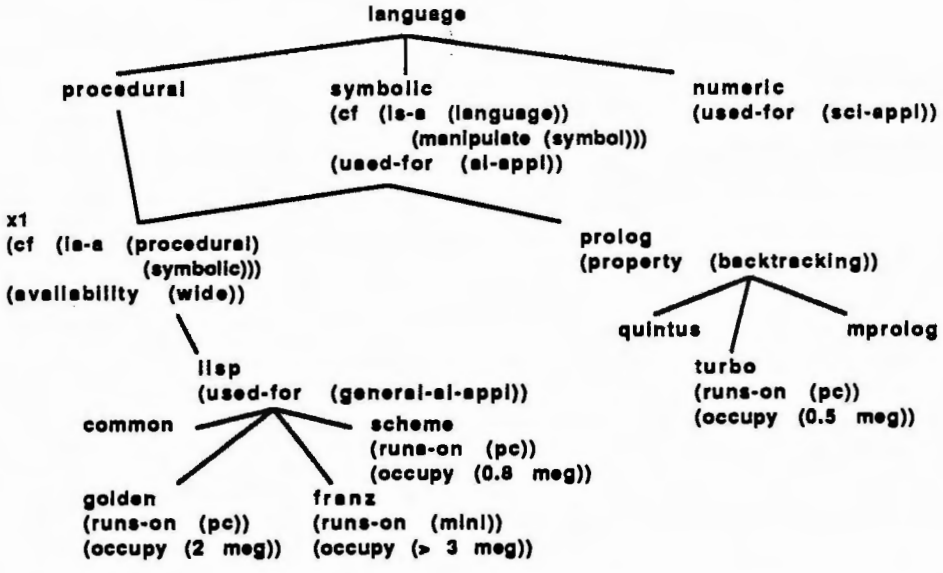


Figure 8: Reorganization of the concepts in LTM after processing the sentence "Lisp is a procedural language."

like this "Lisp is easier to learn than Pascal which in turn is easier to learn than Cobol." In cases like this, we need to ask the expert to rate Lisp, Pascal and Cobol with respect to the predicate *learn*. There are many ways in which this rating can take place and AQUINAS offers several ways which we plan to adopt in our system. Let us consider for the sake of the discussion that the expert produces the following rating: (Lisp (learn .9)) (Pascal (learn (.7)) (Cobol(learn (.4))). Then, the phrase "a symbolic language that is easy to learn" in the user's request "I am interested in a symbolic language which runs in a PC and is easy to learn," will be translated into a numeric value which may be greater than 5 according to the rating provided by the user for that predicate.

7. Discussion

In this paper, we have shown how a consultation system can be built from a natural language description by a human expert. It is our belief that the ideas and techniques described here can be used for the construction of expert systems for tasks which can be modeled using classification problem-solving. These tasks do not have to deal with consultation systems. They may be about diagnosis, design, etc. The relevance of the ideas presented in this paper go beyond the knowledge-acquisition area, since we have introduced a problem-solver which has striking differences from existing rule-based systems. In our system, organization and reorganization of memory, comprehension and problem solving are integrated processes. As a result, two important aspects of expert system technology become almost trivial: the updating of the knowledge base and interacting with the expert system. However, our approach is not without limitations. Let us first discuss the theoretical limitations. Although we believe that a

large class of problems and domains can be handled using our system, there are many domains which are beyond the reach of our method. The reason for that is that there are many aspects of the real world which can not be represented using the classification hierarchies we have presented in this paper. For instance, the functioning of complex systems such as the circulatory system, or the respiratory systems can not be adequately grasped using our present representation techniques. We are working in knowledge representation structures which will represent these systems from a point of view which will be adequate for comprehension and problem-solving (Gomez and Segami, 1989c).

There are many limitations to our present implementation. First of all, no expert has used yet our program to build a consulation system. All the consultation examples have been entered by graduate students or by the implementers of the system. Our system is a prototype which we can "demo" at any time, but that has not been used by any real user. This is clearly in contrast with AQUINAS which, according to the authors, has been used to build a variety of consultation systems. We are going to start to experiment with some real users to assess the real world capabilities of our system. Since SNOWY is able to acquire new concepts and new words, it can augment its lexicon by interacting with a user. The parser (Gomez, 1988) being used by SNOWY is a modular algorithm based on the notion of *syntactical usage of a word*, which allows a naive user without knowledge of English grammar to increase its syntactical coverage. We are writing an interface to our parser which will allow a user to add new verbs not presently in the parser. Yet, we are aware that there is a qualitative jump from testing a prototype by people who have certain familiarity with it to a real world system used by naive users.

We have presented a system and distinguished it from existing knowledge acquisition systems. Yet, our approach can be integrated with present knowledge acquisition tools and viceversa, techniques in other systems can be incorporated in our system. We think that, in future developments of our system for real users, the elicitation techniques developed for ETS and AQUINAS can be very useful. In particular, we are thinking that in those cases in which the English of certain users becomes too hard to handle, SNOWY may fall back on elicitation techniques. Likewise, domain dependent knowledge acquisition tools can benefit from a system which is able to build classification hierarchies from natural language text.

References

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

Boose, J. and Bradshaw, J. (1987). Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for expert systems. *International Journal of Man-Machine Studies*, **26**(1), 3-28.

Charniak, E. and McDermott, D. (1984). *Introduction to Artificial Intelligence.* Addison-WesleyR, Reading, MA.

Clancey, W. J. (1985). Heuristic Classification, *Artificial Intelligence*, 27, 289-350.

Eshelman, L., Ehret, D., McDermott, J. and Tan, M. (1987). MOLE: a tenacious knowledge acquisition tool. *International Journal of Man-Machine Studies*, 26(1), 41-54.

Gomez, F. (1982). Towards a Theory of Comprehension of Declarative Texts, in *Proceedings of the 20th Meeting of the Association of Computational Linguistics*, Toronto, Canada, 36-43.

Gomez, F. (1985). A Model of Comprehension of Elementary Scientific Texts, in *Proceedings of the Workshop on Theoretical Approaches to Natural Language Understanding*, Halifax, Nova Scotia, 70-81.

Gomez, F. (1986). Knowledge Representation for Understanding Expository Texts, Technical Report CS-TR-86, Department of Computer Science, University of Central Florida, Orlando, Florida.

Gomez, F. (1988). WUP: A Parser Based on Word Usage, in *Proceedings of the 1988 IEEE Conference on Computers and Communication*, Scottsdale, Arizona, 445-449.

Gomez, F. and Chandrasekaran, B. (1984). Knowledge Organization and Distribution for Medical Diagnosis, in W. Clancey and E. Shortliffe (Eds.), *Readings in Medical Artificial Intelligence*. Addison-Wesley, Reading, MA.

Gomez, F. and Segami, C. (1989a). The Recognition and Classification of Concepts in Understanding Scientific Texts. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 1, 51-77.

Gomez, F. and Segami, C. (1989b). Classification-Based Inferences in Retrieving Information from a Database of Scientific Facts. In *Proceedings of the Fifth IEEE Conference on Artificial Intelligence Applications*, Miami, Florida.

Gomez, F. and Segami, C. (1989c). Representing Systems as Event Hierarchies. Technical Report CS-TR-89-02, Department of Computer Science, University of Central Florida, Orlando, Florida.

Haas, N. and Hendrix, G. (1980). An Approach to Acquiring and Applying Knowledge. *Proceedings of the National Conference on Artificial Intelligence*, Stanford, CA, 235-239.

Kahn, G., Nowlan, S. and McDermott, J. (1985). MORE: an intelligent knowledge acquisition tool. *Proceedings of Ninth International Conference on Artificial Intelligence*, Los Angeles, California.

Marcus, M. (1987). Taking backtracking with a grain of SALT. *International Journal of Man-Machine Studies*, 26(4), 383-398.

Szolovits, P. and Pauker, S. (1978) Categorical and Probabilistic Reasoning in Medical Diagnosis. *Artificial Intelligence*, Vol. 11.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Knowledge Dependencies
(or What My Ontology Never Told Me About Knowledge Acquisition)

Chris Jacobson
Mike Freiling
Tektronix Laboratories, Tektronix, Inc., P.O. Box 500, Beaverton, OR 97077

ABSTRACT

Several researchers have advocated the construction of high-level "knowledge editors" to assist task experts in developing knowledge bases. In order for the use of knowledge editors to become viable for commercial use, however, the expense in creating such tools must be drastically reduced. In this paper we explore some technical requirements for automating the process of constructing knowledge editors. Utilizing the results of a formal analysis of the task domain and artifacts of the inference engine design, the editor designer then needs to specify knowledge dependencies in the form of invariant relationships between knowledge structures to be acquired and the operational behavior of the interface. Building on a blackboard architecture, this specification should be adequate to generate sophisticated knowledge editors. We illustrate this thesis by analyzing the knowledge dependencies employed in ASTEK, a tool for acquiring knowledge about troubleshooting electronic systems.

1.0 INTRODUCTION

Several researchers have advocated the construction of high-level "knowledge editors" to assist task experts in developing knowledge bases. Musen (Musen, et al., 1986) developed the OPAL system for medical doctors to describe oncology protocols. Jacobson and Freiling (Jacobson and Freiling, 1988) have developed ASTEK for the acquisition of diagnostic troubleshooting knowledge. Knowledge editors typically consist of a collection of windows, menus, graphs, tables, etc. for entering knowledge. We call these mechanisms *media of articulation*, because they provide the medium in which experts must articulate the knowledge that is to be codified as a knowledge base.

These systems are typically custom constructed as a component of a larger knowledge-based system application project. This has resulted in some effective tools for acquiring knowledge within a specific application area. For these tools to gain broader and more

cost effective utilization, the cost of applying the editing mechanisms in a larger number of situations must be greatly reduced.

One approach taken is to develop model-based engineering tools to speed the development of specific types of applications. An example of this is Musen's PROTOGE (Musen, 1988), a tool for a knowledge engineer to generate an OPAL-type knowledge acquisition tool. A similar task aimed at expert system development is Bennett's ROGET (Bennett, 1985), a tool for developing the conceptual structure of expert systems and translating them into parameter and context definitions for EMYCIN. When a single domain is exploited, implicit dependencies between fragments of knowledge can be treated in an ad hoc manner. Unlike these systems, however, we are not working within a single application area, such as clinical advice. We would like to generate knowledge acquisition tools for such disparate applications as diagnostic troubleshooting of electronic systems, process control systems for manufacturing, and rule checkers for design applications. Our initial step in designing a such a tool has been to formulate the requirements for a declarative language that would support describing the knowledge and operational requirements for a particular application. In brief, the language must support descriptions of the following types of information:

(1) the basic knowledge components or *ontology* of the knowledge base.

(2) the appropriate *medium of articulation* for each element in the ontology.

(3) the *knowledge dependencies*[1], invariant relationships between knowledge components and the sequence of activities that comprise the knowledge acquisition process.

(4) the *rules of behavior* for the interface that result from these dependencies

For this paper we have done an analysis of the hand-crafted ASTEK system to identify the necessary components of an acquisition tool description language. We will especially focus on the knowledge dependencies, and the way they have been exploited in ASTEK to support discourse level activities. Appendices A and B present a simplified catalog of the knowledge components and invariants that appeared in the original version of ASTEK.

We believe that creation of a declarative language for describing knowledge components

[1] The terms "dependency" and "invariant" are used somewhat interchangeably in this paper. The general notion is that the well-formedness of the knowledge base depends on specified relationships between knowledge components being held invariant.

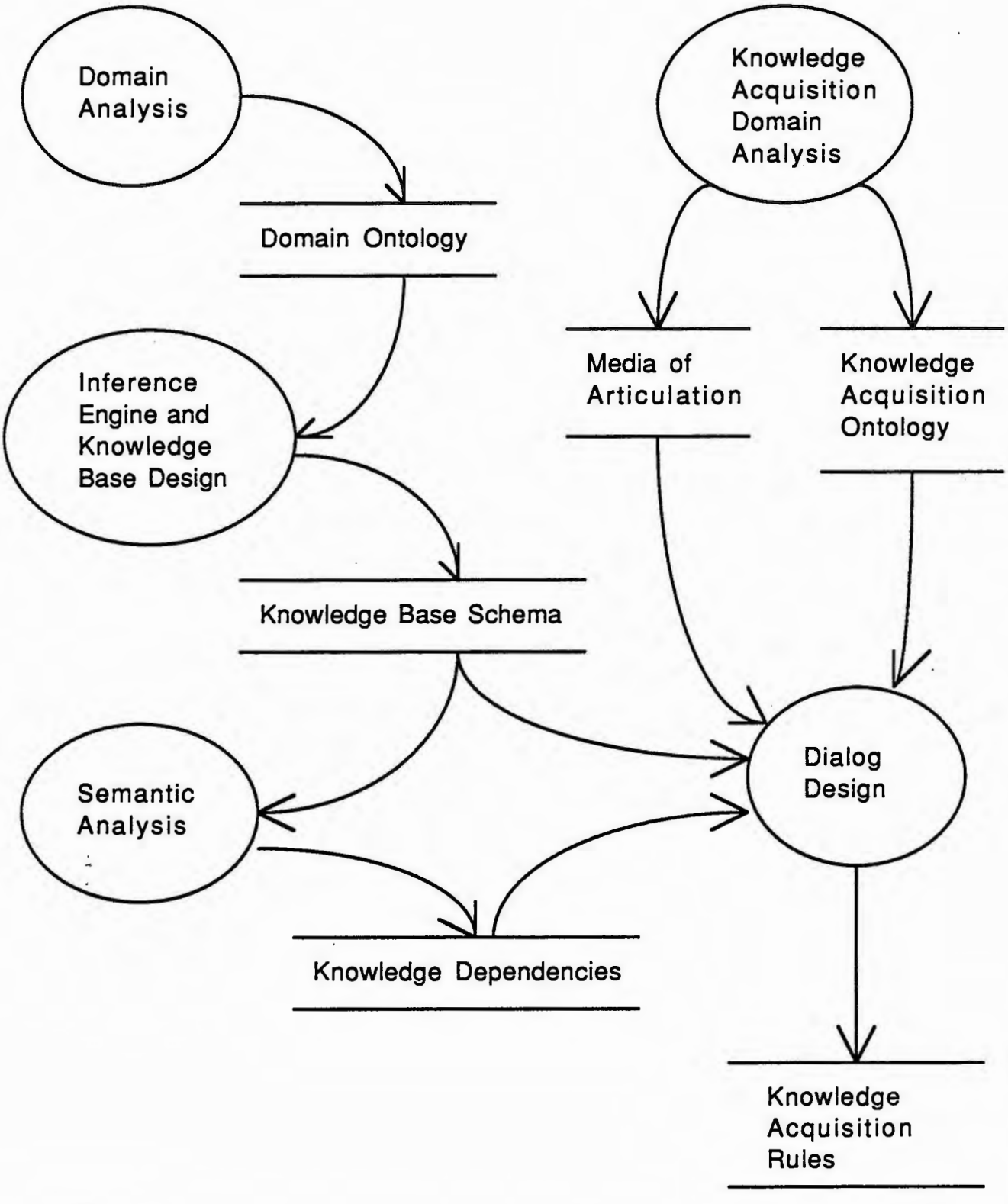


FIGURE 1

An overview: the evolution of the acquisition tool description

and their operational implications is a useful first step on the way to a flexible, domain-independent knowledge editing tool. We will further explore the sources for the tool description information within our methodology for developing knowledge-based systems (an overview of these sources is presented in Figure 1).

2.0 DESCRIBING THE KNOWLEDGE COMPONENTS

Our work on knowledge editors assumes that, at the editing level, all knowledge is organized into frames, which form associations between knowledge pertinent to the task domain (such as a troubleshooting procedure and the diagnostic implications to be drawn from the procedure) with knowledge pertinent to the engineering and maintenance of knowledge bases (such as the author of the knowledge and the date the knowledge was created) and with artifacts of the inference engine design (such as the use of parameters in diagnostic procedures). This is represented in Figure 1 as the transition, by virtue of the design of the inference engine, from a *domain ontology* to a *knowledge base schema.*

Describing the structure of knowledge components requires a language that permits the author to specify what knowledge frames are required, what slots each knowledge frame might contain, and what type of value is expected for each slot. These descriptions share much in common with the notion of a *domain ontology*, which was developed in (Freiling, et al., 1986), (Alexander, et al., 1986), and (Alexander, et al., 1987). In that work, knowledge components were defined as constructions over abstract mathematical domains, using domain combination operators like cross product, union, and function construction. Here, we adopt a language that is slightly more concrete. Instead of abstract domain constructions, knowledge components are identified as either frames, tables, or expressions. This language allows information to be expressed which is not appropriate in a domain analysis (such as slot names for the knowledge structures, information necessitated by the inference engine design, and predicates that are able to restrict the allowable values of a slot).

Frames and tables are used to capture and store major knowledge components. A frame is a collection of slots, whose values can point to other frames or to expressions. A table is described as if it consisted of a collection of slots, but in fact is organized as a collection of records, with each record having a complete set of values for the slots indicated. In some sense, a table represents a single locus for aggregating an entire set of frames. Defining a frame type or table type requires specifying the slots (and their types) that make up the frame or table. Examples from the appendix are the frames STRATEGY and TEST, and the table MODULE_LOCATIONS.

Expressions are used to describe knowledge components that are not important enough to have an independent existence or whose components are not meaningfully manipulated individually. An expression instance only exists as long as it is the value of some slot in a frame or table. We have adopted the language of domain equations that we used for ontological analysis to define expressions, since it provides a concise notation for the essential syntactic structure of each expression. An example from Appendix A is the expression ACTION and its constituents.

3.0 DESCRIBING THE MEDIA OF ARTICULATION

As originally constructed, ASTEK <Jacobson, 1988,ASTEK> organizes its knowledge into frames and slots. To facilitate the knowledge editing task, editors can be individually associated with each slot. When the author selects a slot, the appropriate editor is invoked allowing him to add or edit the value of the slot. We refer to the editors as *media of articulation.* Our research into knowledge acquisition has indicated that task experts prefer different media for different types of editing tasks.

The original ASTEK had implemented five different media of articulation: menu selection, text edting, natural language expressions, graph editing, and table editing. Though a given slot might be describable by more than one medium, the following media tend to be used by slots with the following characteristics:

menu selection: a slot for which one value, selected from a finite number of atomic values, is acceptable. Examples include the inertia and completeness slots of the STRATEGY frame, and the confidence slot of the BELIEF frame.

text editing: a slot for which the value is a generic value like STRING, SYMBOL, or NUMBER and is not intended to be "understood" by the acquisition tool or the inference engine.

natural language expressions: If the type of a slot is well-structured and complex, it is best to edit the value as an independent frame. However, this is unnecessarily cumbersome in cases where the value is not overly complex, as for instance, actions and dates. In such cases, a natural language parser is invoked with a grammar for the appropriate value. In addition, a relatively simple expression occasionally leads to the creation of several frames. For instance, natural language parsing is used to capture expressions like "measure the modem using getBaudRate" which will potentially cause

creation of both a module frame for the modem and a measurement frame for getBaudRate.

Each use of a natural language medium must have associated with it a grammar, a set of interpretations, and a set of rules defining what to do with the resulting interpretations. It is beyond the scope of this paper to go into detail on this area.

graph editing: slots that are filled by graph structures. Such graph structures are very difficult to describe using a linear means of expression such as natural language. In ASTEK examples include the model slot in a KNOWLEDGE_BASE and the procedure and implications slots in a TEST.

Some uses of the graph editing medium require restrictions to be specified on the form of the graphs that are to be allowed (e.g. only trees). The specification of such restrictions is beyond the scope of this paper.

table editing: a table. The table MODULE_LOCATIONS provides an example.

These editing mechanisms should be treated as raw materials for the construction of new acquisition interfaces. For each type of knowledge structure to be acquired, an editing mechanism is associated with each type of slot. This is done with the use of specialized knowledge acquisition rules such as the following simplified example:

```
rule::
    when: add_value (X, procedure, test)
    invoke: graph_editor
```

4.0 DESCRIBING THE KNOWLEDGE DEPENDENCIES

Our initial design for ASTEK provided only the ability to specify knowledge structures and associate with them a variety of editing mechanisms as appropriate. Filling in the skeletal acquisition system with these specifications provided a coherent underlying framework in which to describe complex and varied forms of information. We found, however, that these specifications alone were not adequate to describe the kind of feel that was eventually crafted into ASTEK. We analyzed the system to determine what other information was added to it to give it this feel, a significant part of which we have characterized as knowledge dependencies. In Figure 1 this is referred to as a *semantic analysis* of the knowledge base schema. Though the process of performing this analysis is not yet well understood, we will discuss the resulting product.

Knowledge dependencies are invariants, relationships that are required to hold between different knowledge components. Usually such invariants are required to assure the integrity of the knowledge base, or coherent behavior on the part of the inference engine processing the knowledge base. An example may be found in the domain of electronic instrument troubleshooting. For a completely specified set of diagnostic procedures for an instrument, each module in the instrument must have some troubleshooting knowledge that determines whether it is behaving well or is broken.

Appendix B contains an informal catalog of the knowledge dependencies we have identified in ASTEK. The invariants we have identified so far fall into the following broad classes:

type refinements: Often, the value for a slot in some frame is must be a structure whose components are restricted to certain types. For example, the procedure tree in a TEST may contain only measurement or calculation nodes, while the procedure tree in a MEASUREMENT may also include actions.

inclusion requirements: Several of the ASTEK invariants require that knowledge components referred to in one structure are also referred to in another. Usually this includes one of the "global" structures of ASTEK, the LIBRARY (which contains a sequence of all tests, measurements, and strategies, ordered by recency of reference) and the MODEL (which organizes all parts of the device under test hierarchically). For instance, dependency MODEL(1) states that any module described as the focus of a test or measurement must be included at some level in the MODEL.

default values: Some invariants are not rigid, but rather specify certain default values based on structural considerations. For example, dependency TEST(1) states that the author of a test is inherited from the default author of the knowledge base. It is worth noting that these "inheritance" relationships are somewhat disorderly: they do not follow any clear generalization hierarchy.

semantic reference restrictions: Some invariants restrict the particular instances of components that can be referred to, based on the dynamic structure into which the knowledge fits. The most common variety of reference restriction in ASTEK is the "upstream dependency", of which CONDITION(1), CONDITION(2), and SOURCE_DESCRIPTOR(3) are examples. These invariants all deal with the fact that a parameter referred to in the context of a test (e.g. "if the value of BAUD is less than 1200") must be a parameter of an "upstream" procedure, that is, a procedure that must already have occurred by the time the condition is evaluated.

Another type of reference restriction is found in PROCEDURE_NODE(1), where binding to the parameters of a test procedure can only take place for test procedures that are within the scope of the variable bound.

sequence requirements: Some knowledge components derive their ordering from extraneous considerations. Examples are LIBRARY(1) and LIBRARY(2), which stipulate that the order in which tests and measurements are presented should be determined by recency of use.

5.0 DESCRIBING THE KNOWLEDGE ACQUISITION RULES

The knowledge dependencies we discussed in the previous section are basically to define the well-formedness of the knowledge base as a whole. They arise from the semantics used to model the task domain, and are often required to insure reasonable behavior by the run-time "inference engine."

In addition to their importance in enforcing integrity of the knowledge base, knowledge dependencies have an operational significance in that they can be exploited to reduce the effort required in building a knowledge base. For example, the "upstream dependencies" of ASTEK, such as CONDITION(1) and CONDITION(2) specify that the correctness of parameters which occur in a test condition depends on the parameters being input parameters to the test itself, or be the results of a measurement in the test which occurs prior to evaluating the test condition. Invariants of this type make it possible to construct a relatively short menu of possible parameters to reference in describing a test condition as well as several other operational behaviors in the interface.

It follows that all violations of the invariants must be removed from the knowledge base before using it to perform a diagnosis, just as syntactic errors must be removed from a high-level language program before it can be compiled. However, it is a matter of design choice to determine whether a knowledge dependency be enforced immediately upon detection of the violation, or placed on an agenda for later (but eventual) resolution.

Furthermore, some of the invariants we have seen reflect that a correct value of a slot is dependent on a collection of permissible values (i.e. we have the ability to compute the precise values of slots in the knowledge base, subsets of slot values, or supersets of slot values). The tool designer has the choice of pragmatic uses of such invariants. If a single value is determined, for instance, the slot may be filled automatically. If the dependency only constrains choice to a set of values, the set can be used to construct a menu to guarantee that whatever value is selected be legal.

5.1 A Task Analysis of ASTEK

We analyzed the activity of ASTEK in order to create a model that describes the actions that comprise the acquisition task. This is characterized in Figure 1 as the knowledge acquisition domain analysis resulting in the development of the knowledge acquisition ontology. A portion of the ontology is presented in Appendix C.

We view the items in the resulting knowledge acquisition ontology as falling into two categories. The first is the set of atomic actions that can be performed on the knowledge base. This includes creating and destroying frames, adding and modifying slot values, and creating and editing tables. The second is a set of meta-actions; performing an action, preventing an intended action from being performed, and restricting the behavior of an action. The actions in this ontology will be used to specify the knowledge acquisition rules.

5.2 The Knowledge Acquisition Rules

The task analysis in Appendix C specifies what events can occur in an ASTEK session from the perspective of the user. To fully specify how dependencies are exploited, we need to define a separate concept to specify the single operational implication of a particular instance. These take the form of rules. In order to define these rules, we need to catalog the *actions* that can trigger enforcement of an invariant, and the *effects* that can occur as a result.

Defining the effects of actions is done in two stages. First, we define an **action** as one of the following:

```
add_value( Value, Slot, Frame)
remove_value(Value, Slot, Frame)
```

In other words, all effects of the knowledge dependencies are characterized with respect to their effects on adding or removing values from slots.

Second, we define an **effect** as one of the following:

```
perform(Action)
prevent(Action)
restrict_choice(ChoiceSet, Action)
```

In principle, this defines a wide range of possible combinations. In practice, however, we have observed a few specialized classes of operational uses:

integrity preservation: invariants of this type are checked when an action is attempted. Their effect, if any, is to prevent the action from occurring. Example uses are to prevent removing a test from the library if it is referred to elsewhere in the knowledge base, or to prevent removing a module from the model as long as some test exists for it.

As an example, dependency MODULE(1) states that no two modules can have a parameter of the same name. We can define an integrity preservation constraint as follows:

```
rule::
        when: add_value (X, param, Y: MODULE)
        check: MODULE(1)
        if_violated:  prevent (add_value (X, param, Y))
```

value computation: Invariants of this type are checked when one value is added, and cause computation of other values. A rule associated with dependency SOURCE_DESCRIPTOR(1), for instance, might add a value of nil to the node_slot when a constant is added as the value of the reference slot (i.e. the source is not the result of an operation in an earlier node of the procedure tree).

value restriction: Invariants of this type are checked when the user is about to select some word in a natural language expression. The effect is to compute a menu from which legal selections may be made.

It is worth noting that a dependency can give rise to more than one rule. The most common examples of this phenomenon occur when some dependency specifies non-default (i.e. "hard") values for other slots. One rule is formulated to handle adding the values. A second prevents those values from being removed.

An example is the rule TEST(3), which states that if a module parameter occurs in the device_args of some test, all parameters of modules highest in the model must also occur[2]. The two constraints (the first being a value computation and the second being

[2] Note that each node of the model represents a module (physical or abstract) and any node below it represents a part of that module. Modules are in fact module classes; a parameter for a module contains information specifying an instance of that module class. The values of the parameters of all modules from a given module to the root of the model

a value restriction) formed from this invariant are:

rule::
when: add_value (X, device_args, Y: TEST or MEASUREMENT)
check: TEST(3)
if_violated: perform (add_values (R, device_args, X))

rule::
when: remove_value (X, device_args, Y: MODULE or TEST)
check: TEST(3)
if_violated: prevent (remove_value (X, device_args, Y))

5.3 Rules and the Blackboard Architecture

We will mention in passing that the knowledge acquisition interfaces that we generate would be well implemented through the use of a blackboard architecture (Hayes-Roth, 1984; Nii, 1986). The knowledge structures will reside on the global blackboard as well as the components of the user interface. The knowledge acquisition rules will be implemented as specialized knowledge sources operating on the blackboard. A control knowledge source would be responsible for monitoring the user activity and to alert the appropriate knowledge source when a relevant action is being requested.

6.0 SUMMARY

As was noted by Bennett (Bennett, 1985), knowledge engineers may benefit from the application of knowledge engineering techniques to their own particular talents. We have atttempted to analyze the construction of knowledge acquisition interfaces with the goal of developing tools and methodologies aimed at reducing the time spent developing such interfaces and of making explicit the knowledge required.

In addition to the forms of knowledge that we had addressed earlier, the ontology and the media of articulation, we have identified dependencies between knowledge components in the form of invariant relationships and some corresponding operational behaviors that may be derived from the invariants. We believe that this will allow much of the procedural code present in custom knowledge acquisition systems to be automatically created from a declarative representation of the goals and behaviors of the systems.

As with our approach to knowledge-based system development in general, we have turned

are necessary to uniquely identify an instance of that module.

an eye to the knowledge-engineering process of constructing acquisition interfaces and identified the most error-prone, time-consuming, and least satisfying portion of the task. We are constructing tools and methodologies that will allow the knowledge engineer to focus less on the system coding and more on the application and domain level analysis and specification.

APPENDIX A:
Specification of ASTEK Knowledge Components

```
frame: DISCOURSE_MANAGER
    default_author:  SYMBOL
    kb: KNOWLEDGE_BASE

frame: KNOWLEDGE_BASE
    model: MODEL
    library:  LIBRARY
    databases: TABLE*

frame: LIBRARY
    components: (TEST + MEASUREMENT + STRATEGY) *

frame: MODEL =
    a TREE {T}
      where  [ all NODE in T {N} | structure (N) : MODULE ]

frame: TREE
    root: NODE

frame: NODE
    structure: STRUCTURE
    next_nodes: NODE *

frame: MODULE
    name: SYMBOL
    param: PARAMETER
    keywords: RANGE + SYMBOL *

table: MODULE_LOCATIONS
    module_name: SYMBOL
```

```
    module_location: MODULE_LOCATION_DESCRIPTION

frame: STRATEGY
    name: SYMBOL
    applies_to: MODULE
    device_args: MODULE_PARAMETER *
    procedure:
        a TREE {T} where
            [ all NODE in* T {N} | N: PROCEDURE_NODE and structure(N) : TEST ]
    inertia: { total, strong, medium, weak, none }
    completeness: { total, strong, medium, weak, none }

frame: TEST
    name: SYMBOL
    author: SYMBOL
    applies_to: MODULE
    device_args: MODULE_PARAMETER *
    procedure:
        a TREE {T} where
            [ all NODE in* T {N} |
              N : PROCEDURE_NODE and  structure(N) : MEASUREMENT or CALCULATION ]
    implications: a TREE {T} where
            [ all NODE in* T {N} | N: CONDITION_NODE ]


frame: MEASUREMENT
    name: SYMBOL
    device_args: MODULE_PARAMETER *
    result_args: OUTPUT_PARAMETER *
    applies_to: MODULE
    procedure:
        a TREE {T} where
            [ all NODE in* T {N} |
              N: ( PROCEDURE_NODE or CONDITION_NODE ) and
                  structure(N) : ACTION + MEASUREMENT + CALCULATION]

expression: ACTION = ASK + SEND + LOOKUP + TELL

expression: ASK = {ask_user} x MESSAGE x RESULT
```

```
expression: RESULT = VARIABLE

expression: VARIABLE = SYMBOL

expression: SEND = {transmit} x DESTINATION x MESSAGE

expression: DESTINATION = SYMBOL

expression: MESSAGE = STRING

expression: LOOKUP = {look_up} x TABLE x KEYS

expression: KEYS =
    A sequence of appropriate identifiers for each key slot in the indicated table.

expression: TELL = {tell_user} x MESSAGE

frame: CALCULATION
    name: SYMBOL
    input_args: INPUT_PARAMETER*
    result_args: OUTPUT_PARAMETER*
    procedure:
        a TREE {T} where
            [ all NODE in* T {N} |
              N: ( PROCEDURE_NODE or CONDITION_NODE ) and
                  structure(N) : ACTION + MEASUREMENT + CALCULATION]

frame: PROCEDURE_NODE
    structure: TEST + MEASUREMENT + ACTION
    bindings: ( PARAMETER x SOURCE_DESCRIPTOR ) *
    next_nodes: PROCEDURE_NODE *

frame: SOURCE_DESCRIPTOR
    node_id: NODE + { nil }
    reference: SYMBOL + NUMBER + PARAMETER

frame: CONDITION_NODE
    structure: CONDITION
    next_nodes:
```

(BELIEF_NODE + CONDITION_NODE) x (BELIEF_NODE + CONDITION_NODE)

expression: CONDITION
    grammar:
        < condition> ::= <variable> <comparator> <variable>
        <condition> ::= <variable> <comparator> <constant>
        <condition> ::= <variable> <comparator> <expression>

frame: BELIEF_NODE
    structure: BELIEF
    next_nodes: BELIEF_NODE + {nil}

frame: BELIEF
    module: MODULE
    state: { good, bad }
    confidence: { certain, strong, medium, weak, none }

frame: PARAMETER
    name: SYMBOL
    type: { input, output, module }

    (* Module parameters are strictly used to specialize the device description that a test or measurement is to be applied to. Input parameters apply only to CALCULATIONS, which, unlike TESTS and MEASUREMENTS, do not apply to specific devices. *)

APPENDIX B:
Specification of ASTEK Knowledge Dependencies

The ASTEK knowledge dependencies are described informally in order that they might be more readily understood.

LIBRARY:

(1) The component sequence for the library is reordered in most-recently-used order whenever a frame is explicitly accessed.

(2) The sequence order of components in the library should be represented in menus that are presented for selecting tests and measurements in procedure trees. (At present, this invariant is not actually implemented in ASTEK, but should be implemented in the next version.)

(3) Every STRATEGY, TEST, or MEASUREMENT that appears in any slot in the knowledge base must also appear in the library.

MODEL:

(1) Any module in the applies_to slot of a TEST or MEASUREMENT must exist as a structure of some NODE in the MODEL.

MODULE:

(1) No two modules can have a parameter of the same name.

TEST:

(1) The value of the author slot is inherited as a default from the default_author slot in the DISCOURSE_MANAGER.

(2) Any MEASUREMENT referred to in a TEST procedure tree must appear in the library.

(3) If a module parameter occurs in the device_args slot of a TEST, all module parameters of modules in the MODEL that are higher up must also occur.

PROCEDURE_NODE:

(1) The parameters that appear in the bindings of a PROCEDURE_NODE must be input or module parameters of the subsidiary TEST, MEASUREMENT, or ACTION.

(2) If any of these arguments are also device_args of the immediately enclosing TEST or MEASUREMENT, then bind them to the same value as bound above, but as defaults that may be overridden. (Binding in this context is actually "sharing", to the bound value is stored only once, and the binding in the parameter node points to the parameter cell for the module parameter of the enclosing TEST or MEASUREMENT.)

SOURCE_DESCRIPTOR:

(1) When the reference slot is filled by a symbol or number, the reference is to a constant, so the value for node_id must be nil.

(2) When the reference slot is filled by a pointer to a parameter, a node_id of nil indicates the parameter is an input parameter of the test or measurement whose procedure tree this source descriptor is attached to a node of.

(3) When the reference slot is filled by a pointer to a parameter, a node_id value that is not nil must indicate *another* node (an upstream node, in fact) of the tree to which the node owning this source_descriptor is attached.

CONDITION:

(1) All variables that appear in a CONDITION that is part of the implication of a TEST must appear as input variables of the TEST itself or output variables of some measurement in the test's procedure tree.

(2) All variables that appear in a CONDITION that is part of a MEASUREMENT or TEST procedure must be input variables to the MEASUREMENT or TEST or else output variables from upstream nodes in the procedure tree.

APPENDIX C:
A Domain Analysis of Knowledge Acquisition

This appendix contains a partial ontology of the knowledge acquisition task in ASTEK.

KNOWLEDGE_ACQUISITION_TASK =
    KS_CREATION_TASK +
    KS_DESTRUCTION_TASK +
    KS_EXAMINE_TASK +
    KS_ALTER_TASK +
    NL_EXPRESSION_TASK

KS_TYPE = FRAME_TYPE + TABLE_TYPE

KS_NAME = FRAME_NAME + TABLE_NAME

KS_CREATION_TASK = {create} X KS_TYPE

KS_DESTRUCTION_TASK = {destroy} X KS_NAME

KS_EXAMINE_TASK = {edit, close} X KS_NAME

KS_ALTER_TASK = TABLE_ALTER_TASK + FRAME_ALTER_TASK

FRAME_ALTER_TASK = {add_value, remove_value} X FRAME_NAME X SLOT_NAME X VALUE

TABLE_ALTER_TASK = ROW_ALTER_TASK + COLUMN_ALTER_TASK

ROW_ALTER_TASK = {add_row, remove_row} X TABLE_NAME X ROW_INDEX X ROW_VALUE

COLUMN_ALTER_TASK = {add_column, remove_column} X TABLE_NAME X COLUMN_NAME

NL_EXPRESSION_TASK = {express} X EXPRESSION_NAME X FRAME_NAME X SLOT_NAME

KNOWLEDGE_ACQUISITION_EFFECT =
    ({perform} X KA_ACTION) +
    ({prevent} X KA_ACTION) +
    ({restrict} X FRAME_NAME X SLOT_NAME X VALUE_SET)

KNOWLEDGE_ACQUISITION_RULE =
    ACTION_SEQUENCE_RULE +
    INTEGRITY_PRESERVATION_RULE +
    VALUE_COMPUTATION_RULE +
    VALUE_RESTRICTION_RULE

ACTION_SEQUENCE_RULE =
    (ACTION_TYPE X {must preceed} X ACTION_TYPE) +
    (ACTION_TYPE X {must immediately preceed} X ACTION_TYPE) +
    (ACTION_TYPE X {must preceed} X ACTION_TYPE X {without} X ACTION_TYPE)

INTEGRITY_PRESERVATION_RULE =
    {check} X DEPENDENCY_NAME X {name} X ACTION_TYPE X
    {if violated} X KNOWLEDGE_ACQUISITION_EFFECT

<... and far more than warrants inclusion in this paper>

REFERENCES

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

Alexander, J. H., Freiling, M. J., Shulman, S. J., Rehfuss, S. and Messick, S. L. (1987). Ontological Analysis: An Ongoing Experiment. *International Journal of Man/Machine Studies*. **26**(4): .

Bennett, J. S. (1985). ROGET: A Knowledge-Based System for Acquiring the Conceptual Structure of a Diagnostic Expert System. *Journal of Automated Reasoning*. **1**: 49-74 .

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.In D. Sriram and R. Adey Eds., *Applications of Artificial Intelligence in Engineering Problems*. New York, N.Y., Springer-Verlag.

Hayes-Roth, B. (1984). A Blackboard Model of Control. *Technical Report, HPP 83-38*,Stanford University.

Jacobson, C. and Freiling, M. J. (1988). ASTEK: A Knowledge Acquisition Tool for Complex, Structured Knowledge. *International Journal of Man/Machine Studies*. **29**(3): 311-327 .

Musen, M. A. (1988). Generation of Model-Based Knowledge-Acquisition Tools for Clinical-Trial Advice Systems. *Thesis,* Stanford University.

Musen, M. A., Fagan, L. M., Combs, D. M. and Shortliffe, E. H. (1986). Using a Domain Model to Drive an Interactive Knowledge Editing Tool. *Proceedings of the 1st AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada.

Nii, H. P. (1986). Blackboard Systems. *Technical Report, STAN-CS-86-1123*,Stanford University.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Bringing corporate knowledge into focus with CAMEO™

William P. Jones
Arthur D. Little, Inc.
Acorn Park
Cambridge, MA 02140

CAMEO™ is a versatile knowledge acquisition tool promoting the Computer-Aided Modeling of Expertise in Organizations. CAMEO supplies a critical link between the expert and the expert system that is often missing in knowledge engineering efforts. A CAMEO-produced knowledge model serves as design specification and documentation of the expert system. A CAMEO model can also be converted into the code of an expert system shell and can guide in the selection of a shell with the appropriate functionality. Our experience indicates that knowledge models frequently have considerable additional value in organizations as vehicles of training and communication.

INTRODUCTION

The concept of knowledge management and an appreciation of its importance in the corporate world is a recent phenomenon. Companies see a future of rising training and turnover costs and ever greater scarcities of critical expertise. At the same time, market forces, particularly in service-oriented sectors of the economy, are driving companies to decentralize and distribute many knowledge-intensive activities. Companies are increasingly interested in the management of knowledge as a corporate asset, which, like any other asset, should be accumulated, protected and utilized to maximum benefit.

Corporate interest in knowledge management has led to a considerable investment in efforts to build expert systems. There is, however, a

growing awareness that expert systems development technology is only one piece of the corporate solution to problems of knowledge management. Many expert system projects, for example, have foundered for want of an adequate model of the knowledge that the expert system is intended to implement. The absence of such a model can lead to a "Tower of Babel" effect: the knowledge engineers of a project have increasing difficulty understanding each other's efforts; indeed, an engineer may have difficulty understanding his own code as the development proceeds. The complexity of the system then seems to increase exponentially in relation to its size (e.g., number of rules) so that efforts to modify or extend the system slowly grind to a halt.

A final irony arises even in projects that realize a working, useful system. The attempt to make a given expertise more widely available in the company has produced a new group of experts - the project's knowledge engineers. Woe betide efforts to maintain or update the expert system should these new experts leave the company! The attempt to "open the black box" of the expert's knowledge has, in fact, produced just another black box. The knowledge imbedded in the code of an expert system is no more easily understood (e.g., by company managers, employees in training) than the corresponding knowledge embedded in the mind of the expert. Various desirable conversions of this knowledge, into a training manual or the code of another expert system shell, for example, is often out of the question.

Computer-Aided Modeling of Expertise in Organizations, CAMEO™, is intended to produce knowledge models with a more accessible shell-independent expression of knowledge. A CAMEO-produced knowledge model can be used in a variety of ways:

• A tool of *conceptual debugging* which makes apparent some of the gaps and inconsistencies in existing corporate expressions of knowledge (e.g., expert interviews, written guidelines).

• A *blueprint* for expert system construction - A knowledge model constructed using CAMEO can be used to guide the construction of an expert system. CAMEO is, in fact, able to translate a model into the initial code of any of several expert system shells. In addition, the knowledge model can guide the user towards the selection of a shell with the appropriate functionality for the job at hand.

• *Documentation* of the expert system - The knowledge model supports the system's use and maintainability.

• A vehicle of *corporate communication* - The knowledge model has tremendous general value, independent of its use in expert system building, as a means of accessing, understanding, and documenting the "black box" of an expert's decision-making processes.

These uses of the knowledge model argue for an expansion of Buchanan's definition of knowledge acquisition as *"the transfer and transformation of problem solving expertise from some knowledge source to a program."* (Buchanan, 1983). We measure the generality of a student's acquisition of knowledge by its appropriate application in a variety of situations. Similarly, the success of a knowledge acquisition effort can be measured by the extent to which expertise, possibly from a variety of knowledge sources, is transferred and transformed for a variety of uses. The construction of a program such as an expert system is only one such use.

At the expense of introducing still more jargon into the field, we will refer to CAMEO as a *knowledge modeling system* (KMS). The use of a KMS differs considerably from that of a *knowledge support system* (KSS) with its intent "to automate knowledge engineering as a process of direct interaction between domain experts and the computer" (Shaw & Gaines, 1988). Among the recent KSS development efforts can be included the work of Boose (1984, 1985, 1986), Boose & Bradshaw (1988), Gaines & Shaw (1980), Shaw (1982), Shaw & Gaines (1986,1988), and Wahl (1986).

These systems share a common *repertory grid* approach (Shaw, 1981) inspired by Kelly's development of *personal construct psychology* (Kelly, 1955).

Unfortunately, KSS development efforts are still many years away from realizing a system that eliminates the need for human knowledge engineers. In our work with various companies, the repertory grid approach has been used to some advantage as a means of acquiring a first pass understanding of an expertise. We invariably reach a point, however, where more direct control over the structure of the emerging knowledge model is needed. In a CAMEO-based knowledge modeling effort, the user retains this control while CAMEO assumes much of the lower-level, bookkeeping tedium associated with the construction of a knowledge model. CAMEO also provides a framework to guide users through the higher-level decisions of knowledge modeling.

A KMS such as CAMEO supports a careful, labor-intensive, and sometimes protracted process of knowledge modeling. Although a KMS may be quickly mastered (proficient use of CAMEO is attained after only about 8 to 24 hours of training and practice), the construction of a complete model may take many weeks of thought and effort. Given the time involved, a company will generally designate a special team of knowledge modelers to complete the task. Corporate investment in such efforts can be paid for many times over through dramatically improved success rates of expert system projects. The investment can pay for itself in other ways through reduced costs of employee training and turnover, improved flexibility and precision in the realization of corporate goals, and general improvements in the availability and quality of corporate knowledge.

Desiderata of a Knowledge Modeling System

A KMS should assist in the construction of knowledge models that are:

complete - a KMS should aid in the detection of gaps in the

emerging knowledge model. Is an action, for example, appropriately connected to triggering conditions? Do conditions, directly and indirectly, enable the appropriate actions?

consistent - a KMS should help the user to target possible conflicts in the rule set. Does the same condition combination, for example, trigger incompatible actions through different rules?

concise - a KMS should assist in the minimization of model redundancy. The system should identify rules, for example, with similar actions and similar conditions as candidates for generalization.

clear - a KMS should enable approaches to model construction that comport well with expert representations. In addition, the system should provide a variety of model displays to accommodate different audiences (e.g., experts, trainees, managers, knowledge engineers).

credible - a KMS should promote the ready validation of model components, e.g., through cross referencing to the text of an expert interview and other motivating documentation.

Knowledge Modeling in Organizations

Additional circumstances surrounding an organizational effort to model its knowledge are worthy of mention.

Knowledge in organizations has a variety of sources, including various experts, written guidelines, and common knowledge regarding rules and procedures that may or may not be documented. In addition, knowledge modelers may themselves serve as a knowledge source in order to fill in gaps in a knowledge model. A knowledge modeling system should help to

maintain a mapping between model elements and motivating text that identifies the knowledge source and the date of its elicitation. The system should provide modelers with opportunities to comment on model elements and knowledge sources. The system should also contain tools that help to pinpoint the (currently) relevant information of a knowledge source.

Knowledge modeling/engineering in organizations is a group activity. Companies perceive benefit in the different perspectives that members of a knowledge modeling effort bring to the task. More importantly, team projects are better able to cope with the loss of one or two of its members. On the other hand, the costs of coordination, communication, and consensus-building in a team project can be considerable. Facilities of a knowledge modeling system should help to reduce these costs.

Knowledge engineering skills of team members may be minimal. Team members may have completed short-duration seminars in knowledge engineering, but they seldom have much hands-on experience. They must, therefore, be in a situation where they can "learn as they go". This argues for a KMS with a minimalist approach to knowledge modeling. The system should center on a small number of constructs (e.g., element types and relations) that are easily mastered and yet representationally adequate. The KMS should not burden the user with excessive functionality that calls undo attention to knowledge representation esoterica. The knowledge model should guide the construction of a working expert system, but it does not have to be a working system.

Much of the expertise companies wish to model involves case-based classification. The job of many corporate experts is well-characterized as a classification task. For each incoming case (e.g., loan applications, insurance applications, bug reports) the expert attempts to make an appropriate classification through the action(s) selected, i.e. the case disposition. A life insurance underwriter, for example, decides whether to accept or reject an application or to order one of a small number of

additional information gathering activities (e.g., a medical exam, an inspection, an attending physician statement). Again, this argues for a simple KMS system that can straightforwardly link model elements to case examples.

CAMEO, as a KMS, has been designed and developed in close collaboration with a large insurance company with a view towards these circumstances of the organizational knowledge modeling effort. Members of a knowledge modeling team quickly gain proficiency in the mechanics of CAMEO's use (after 8 to 24 hours of training). CAMEO then plays a critical role in promoting the cooperative work of team members. Presentation modes of CAMEO also help others, outside the initial team, to understand and critique the emerging knowledge model. In this way, CAMEO often helps to preserve an essential involvement of experts and management.

AN OVERVIEW OF CAMEO

System specification

CAMEO is based in HyperCard™[1] and runs on any Macintosh Plus, Macintosh SE, or Macintosh II with a standard configuration of at least one megabyte of memory and a hard drive. CAMEO is written almost entirely in HyperTalk™, the interpretive, English language-like scripting language of HyperCard. Constructs of HyperCard are used in CAMEO to promote a hands-on, "learn as you go" style of interaction.

---

[1] HyperCard is a powerful collection of information management and user-interface management tools. Under HyperCard's object-oriented approach, with its use of graphics-based metaphors, users store information on "cards" which are sequenced together into "stacks". Individual cards have "fields" that store text and "buttons" that can be pressed to accomplish various actions. Buttons can, for example, link cards together in ways that capture important relationships in a body of information.

Recent history

CAMEO has undergone considerable refinement through its daily use at a large insurance company to model knowledge of the company's head life insurance underwriter (who is soon to retire). Three members of a knowledge engineering team separately used CAMEO to analyze transcripts of a weeklong, case-based interview with the expert. Every two to three days members switched to group use of CAMEO in order to update the collective knowledge model through the integration of these individual efforts. This modeling effort, now nearly complete, has resulted in significant modifications of underwriting guidelines. Presentation of the model to management was a great success and has helped to elicit support for a Phase II effort to build an expert system.

Basic constructs of CAMEO

A knowledge model built in CAMEO is one or more HyperCard stacks containing knowledge model elements (one element per card) connected to one another. There are four major element types in CAMEO: *rules*, *assertions*, *attributes*, and *objects*.

Knowledge model elements are interconnected via two basic relations: a vertical relation of *ownership* and a horizontal relation of *entailment*. The ownership relation can be used to structure knowledge model elements into a meaningful taxonomy that facilitates comprehension of and navigation through the knowledge model. Objects can own other objects and attributes. Attributes can own rules and assertions. An object such as "the life insurance application" might own the object "the proposed insured" and the attributes "the date of application" and "the amount of coverage". An attribute such as "the amount of coverage", in turn, can own assertions such as "the amount of coverage is less than $10,000" and it might own a rule such as "if the amount of coverage is less than $10,000 then accept the life insurance application". Rules that do not comfortably fit under an attribute in the model's current ownership hierarchy, can be organized under "dummy" attributes of the user's creation. Rules and

assertions are a model's bedrock and cannot themselves own other elements.

The entailment relation is, of course, central to any knowledge model. CAMEO entailment relationships weave together assertions and rules into a simple and/or graph. A configuration of input assertions (observable conditions) can be conjunctively grouped via a rule element to other assertions which may be internal or output (actions). It may be observed (e.g. from the application) that "the age of the proposed insured is > 60" and that "the medical declarations of the proposed insured are all nos" (i.e., the proposed insured asserts perfect health). The rule representing the thinking of a suspecting underwriter might group these two assertions via entailment relationships to an internal assertion that "medical problems of the proposed insured are possibly hidden". This assertion in combination with others may then be linked via another rule element to an output assertion such as "the disposition of case is to order a medical examination". An arbitrary number of rule elements can feed into an assertion such as this one in a disjunctive entailment relationship, i.e., if any one of these rules is enabled, the assertion is judged to be true.

An illustration of CAMEO's use

Figure 1 displays a card representing a rule element in CAMEO. The icons on the top left- and right-hand sides of the card represent various card and navigational options. Top center in Figure 1 we see the name of this stack - "Elements" in this example. Below the stack name is a brief label for the knowledge model element represented on the card - "hidden=>medExam" in this example.

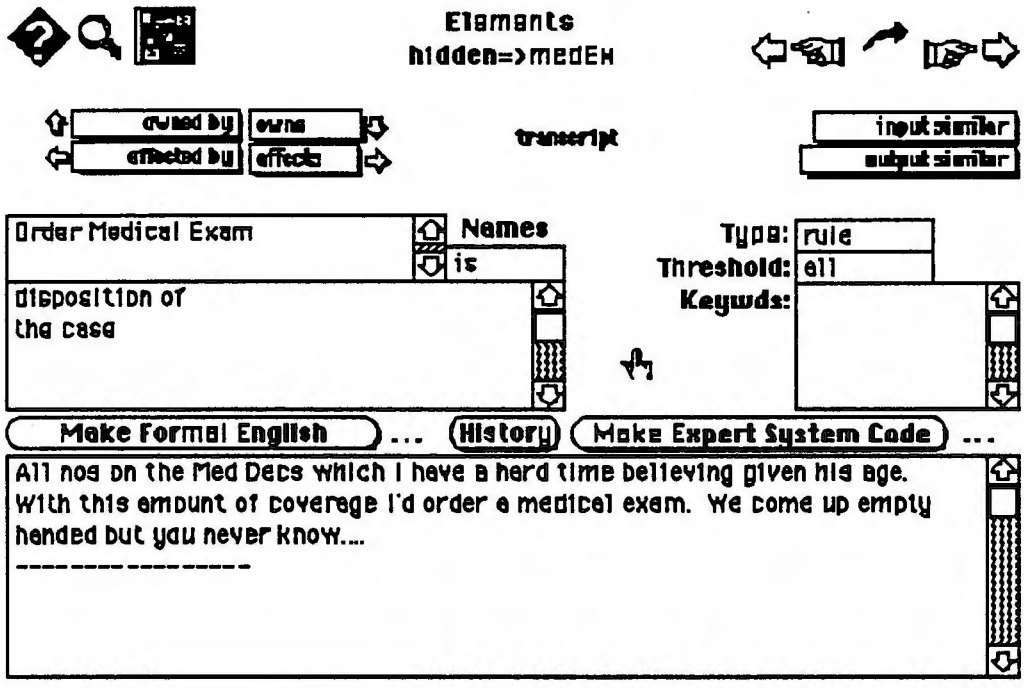


Figure 1. A sample "rule element" card in CAMEO.

Farther down the card of Figure 1 are buttons representing some possible relationships between the card and other cards of the model. A rule element such as that of Figure 1 can be "owned by" an attribute but cannot "own" any other elements. A rule element is "affected by" assertion elements that are the antecedents of its rule and it "affects" other assertion elements that are the consequents of its rule. The "transcript" button represents cross referencing between the element of the card and information sources. The text in the bottom field of Figure 1, "All nos on the Med Decs which I have ...", is an excerpt from an interview with a life insurance underwriter.

Going back to the knowledge source ... Clicking on the "transcript" button takes the user to the more complete text on the card of Figure 2 with the excerpt highlighted within the text. (If the element card of Figure 1 had contained more than one excerpt the user is able to select the desired reference from a menu of alternatives.) CAMEO automates the

input of ascii text and its segmentation into the card-sized chunks (about the length of a paragraph) of a "transcript" stack.

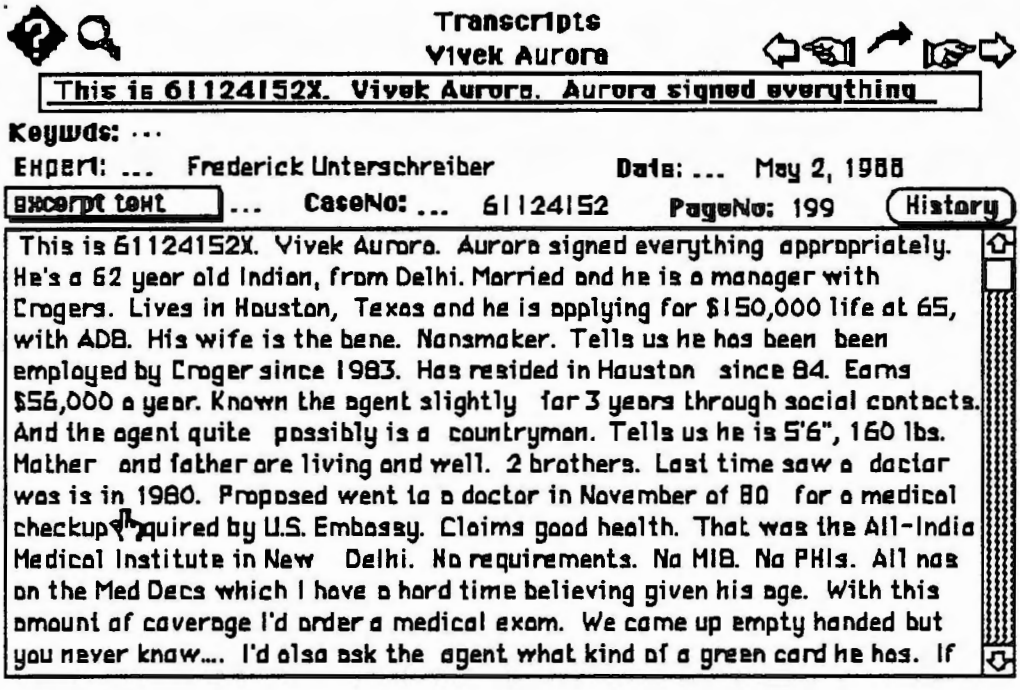


Figure 2. A sample transcript card - the "source" for the card in Figure 1.

CAMEO supports a many-to-many mapping between the cards of such a stack and element cards of the knowledge model: The text card of a transcript stack such as that of Figure 2 may motivate several knowledge model elements; similarly, a knowledge model element may be motivated by several different text cards. At the insurance company, this cross referencing facility quickly answers such questions as "where did this come from?" - whether these questions are posed by management or other members of the knowledge modeling team. It should be noted, however, that use of this facility is optional. Users are free to create and link element cards without ever "grounding" them through linkage to source in a transcript stack.

The match card facility ... The cards of a transcript stack could easily number in the thousands. CAMEO supports the selective access and

analysis of these cards through a *match card* facility. A newly created match card of a transcript stack has the structure of Figure 2 without its content. The user formulates a query by filling in various slots. The match card of Figure 3 indicates a desire to see cards dated from May of 1988, involving "Bob Blah" as the source, and containing either the phrase "Medical Information Bureau" or the acronym "MIB". Within a field disjunctions are signaled by a "|" and conjunctions are signaled by an "&"; the relationship between fields is always conjunctive.

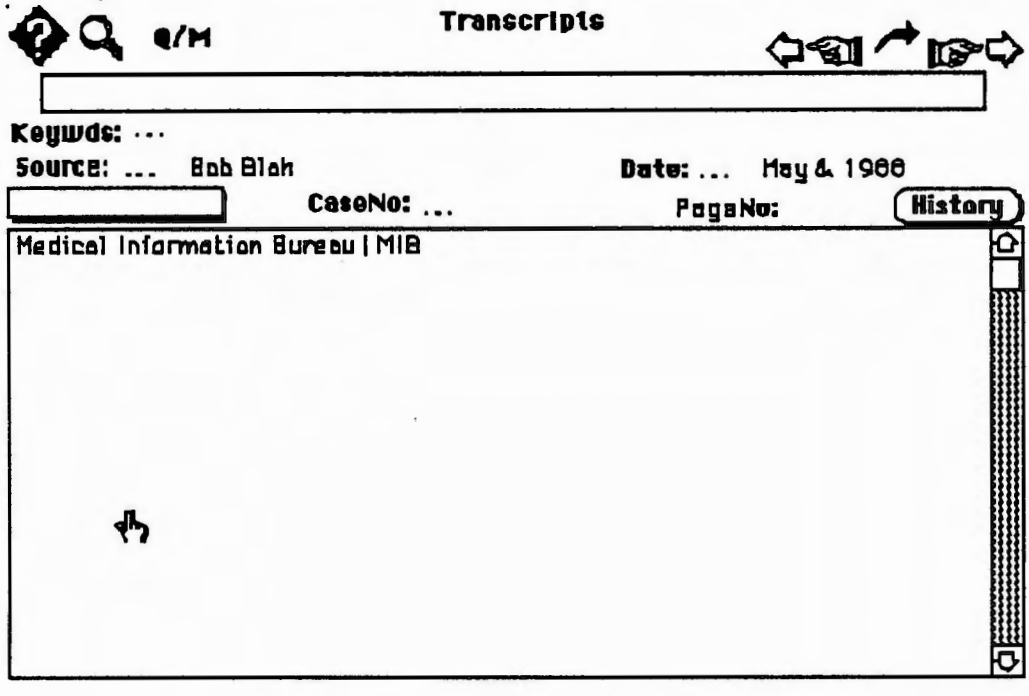


Figure 3. A sample "match card" used to establish card subsets.

A match card is used to generate a listing of cards that meet the match card's constraints. The user can then perform various group operations on these cards or the user can go directly to one of the matching cards of a list. The user can also use the "hand" icons, top right on the cards of Figures 2 and 3, to step forwards or backwards through the cards of a match card listing. The user can create any number of match cards to accomplish desired partitions of a stack's cards. A similar match card facility exists for element stacks containing element cards such as that of

Figure 1. The match card facility is a simple but indispensable means of reducing the sometimes enormous size of a knowledge model to manageable proportions.

The creation of elements and their links ... The user can graph an element's relationships of ownership or entailment. Figure 4 displays current entailment relationships for the element of Figure 1. We can interpret this to represent a rule stating that "if medical problems of the proposed insured are possibly hidden then order a medical exam". By double clicking on the boxes of this display, the user can navigate forwards or backwards through the entailment network. The user might double click on "med exam", for example, to see all rules that trigger the action of ordering a medical exam. In the other direction, the user might learn that the suspicion of hidden medical problems is entailed, via a rule element, when a proposed insured is over 60 and answers the entire medical declaration section of the application in the negative.

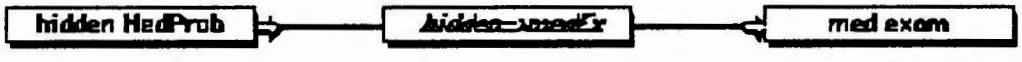


Figure 4. The graph of a rule element's entailment relationships.

In our example, however, we are interested in elaborating on the condition side of the rule represented in Figure 4. The text of Figure 2 leads us to suspect that the underwriter does not order a medical exam solely on the suspicion of hidden medical problems. The underwriter also mentions the amount of coverage requested in the policy. Perhaps, we might reason, medical exams are expensive so that the underwriter orders an exam only when the amount at risk (and the amount to be gained in premiums) is large enough - greater than, say, $100,000. We reflect this by adding another condition to the rule represented in Figure 4.

To do this we return to the card of Figure 1 by clicking on the center box "hidden=>medExam". We then save the location of this card (by double-clicking on the "magnifying glass" button) before setting off in search of a card with the assertion to the effect that the amount of coverage is large. There are a number of ways to find such a card. We might create a match card or visit an existing one. We might move up the ownership hierarchy from the current card. Or we might start from the "world" object (owning, by definition, all other objects in the model) and move down the ownership hierarchy.

Pursuing this last option, we might see the graph of Figure 5 showing ownership relationships of the "Case" object element. The "«more»" box indicates that the "Case" object owns more elements than can be displayed at one time. We can click on this box to see additional elements. However, the case attribute we are interested in, "coverage amount", is already visible. We continue down the ownership hierarchy by double-clicking on this box to see the display of Figure 6.

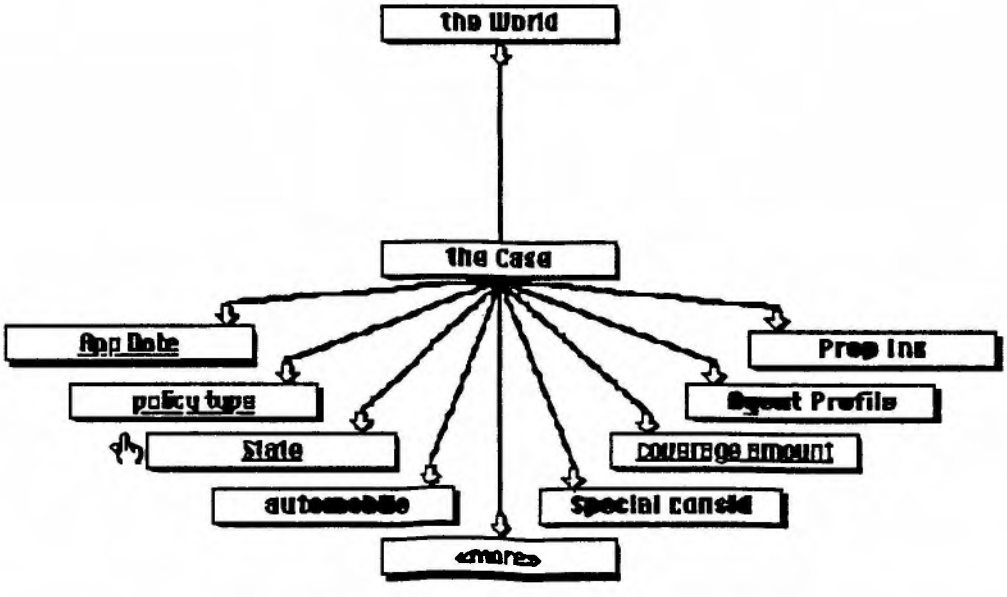


Figure 5. "the Case" element's view of the ownership hierarchy.



Figure 6. The "coverage amount"'s view of the ownership hierarchy.

Our assertion is not there so we simply create it (Figure 7) letting CAMEO lead us through a dialogue to establish its label and value (> $100,000). Note that although our search for the assertion failed, it placed us in the proper position to create this assertion (as a child of the "coverage amount" attribute). Use of the ownership hierarchy, especially in team efforts, helps to prevent the redundant creation of objects, attributes, and assertions.

Now we toggle back to the rule element card, location already saved, by double clicking on the "takeoff arrow" button on the top right-hand side of Figure 7. This also serves to save the current location, the new ">$100,000" assertion. When we now double click on the "affected by" button of Figure 1, CAMEO toggles back to the ">$100,000" assertion to confirm that this is the condition we wish to add. After answering in the affirmative, a new link is added as we see in the newly generated graph of the rule element's entailment relationships of Figure 8.

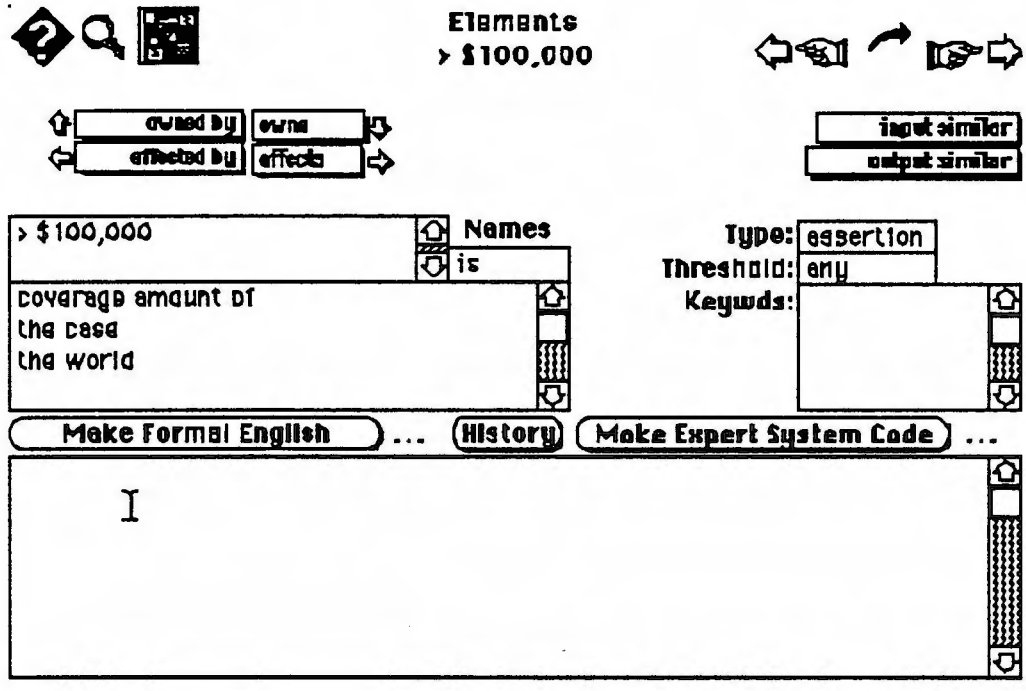


Figure 7. The newly created "> $100,00" assertion.



Figure 8. The new graph of the rule element's entailment relationships.

In our creation and navigation of element links (of ownership and entailment) we may frequently cross stack boundaries (represented as different files in the Macintosh operating system). This process is completely transparent.

Formal English and expert system code ... We can now generate a somewhat stilted expression of our rule by clicking on the "Make Formal English" button of Figure 1. Or we could elect to generate expert system

code for the rule by pressing the "Make Expert System Code" button. Both of these options have been exercised in the display of Figure 9. The lower pop-up field now contains a KEE™ language expression of the rule. CAMEO allows us to comment freely on the rule in its various forms through the use of pop-up fields.

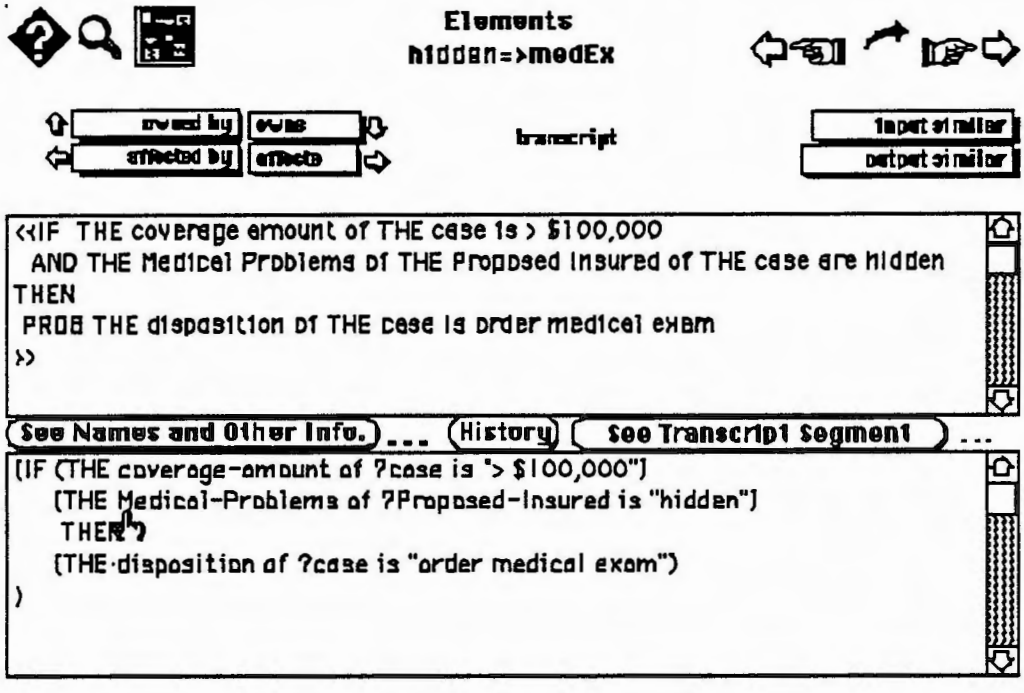


Figure 9. Formal English and KEE™ code representations of the rule element's entailment relationships.

Other CAMEO functions

Model modifications ... Knowledge model elements or their links to other elements can be deleted (following a confirmation safeguard). Elements can also be renamed or repositioned in the organizational hierarchy. CAMEO automatically performs all the requisite bookkeeping associated with these and other modifications of the network.

Consolidation, completeness, and consistency checking ... CAMEO can represent important similarities among model elements through use of

the "input similar" and "output similar" buttons (see Figures 1, 8 or 9). Similarities can be manually noted by the user or automatically generated by CAMEO. We can click on the "input similar" button of Figure 1, for example, to view a list of other rule elements that are "affected by" the assertion elements bearing this relationship to the rule element of Figure 1. Similarly, we can click on the "output similar" button to see rule elements that "affect" the same assertion elements. In both cases, the rule elements of a listing are ordered according to the number of assertion elements they share in common with the rule element under consideration.

Using the "input similar" facility, we are able to identify possible conflicts among the model's rules. When rules with the same antecedents produce incompatible consequents, we can use CAMEO to connect additional assertion elements to rule elements to achieve better discrimination among the model's rules. Through use of the "output similar" facility, we are able to identify opportunities for generalization among the model's rules. Perhaps a single rule, with the right set of antecedent assertions, can replace a set of rules with the same consequents.

CAMEO's "absorb element" function greatly facilitates the process of rule element consolidation, e.g. when this is done to generalize or to eliminate redundancy. Rule elements of a set are "absorbed" into a "surviving" rule element which acquires all entailment links and transcript cross-referencing of the absorbed elements. We can then make the necessary modifications to the surviving rule's relationships of entailment. The "absorb element" function can be used in similar fashion to eliminate redundancy among assertions, attributes, and objects of a model.

Model consolidation and the checking of consistency and completeness are also facilitated by the "match card" facility which has already been discussed. In addition, an "expand" facility has proven very useful. Starting from any model element, we can elect to see a breadth- or depth-first expansion of links in any of four directions - up ("owned by"), down

("owns"), left ("affected by") and right ("affects"). We can, for example, see descendants, direct and indirect, of "the Case" object (Figure 10). Or we can see all effects, direct and indirect, of the "> $100,000" input assertion. Or we can see all circumstances in which a medical exam can be ordered. The expand facility is routinely used in underwriting project to reveal the current state of a model's interconnectedness and to understand the effects of proposed modifications to the model.

Group work and the inter-stack transfer of element cards ... The design of CAMEO is oriented towards the team work of an organizational setting. Using CAMEO, the members of the team will alternate between individual and group activity. In the life insurance underwriting project, the ascii text of expert interviews was input by CAMEO into different transcript stacks which were then divided among team members. Each team member worked individually through his assigned transcript stacks, using CAMEO functions (esp. the match card facility) to identify important references to rules and assertions (e.g., observations, actions). These could then be excerpted and used as a basis for the creation of model elements.

Every two or three days team members met to consolidate their efforts and to elaborate an emerging "world view" stack of objects, attributes, and key assertions. In the course of specifying the entailment relationships, team members would individually add object, attribute and assertion cards to their versions of the ownership hierarchy. During the group consolidation effort, these additions were reviewed and checked for wording and possible redundancy. Element cards remaining after the completion of name changes and "absorb element" and "delete element" operations could then be moved via CAMEO's "move elements" facility to an updated "world view" stack. Team members copied this stack to their private, working directories before returning to individual work. All bookkeeping associated with the modification and transfer of model elements is automatically handled by CAMEO.

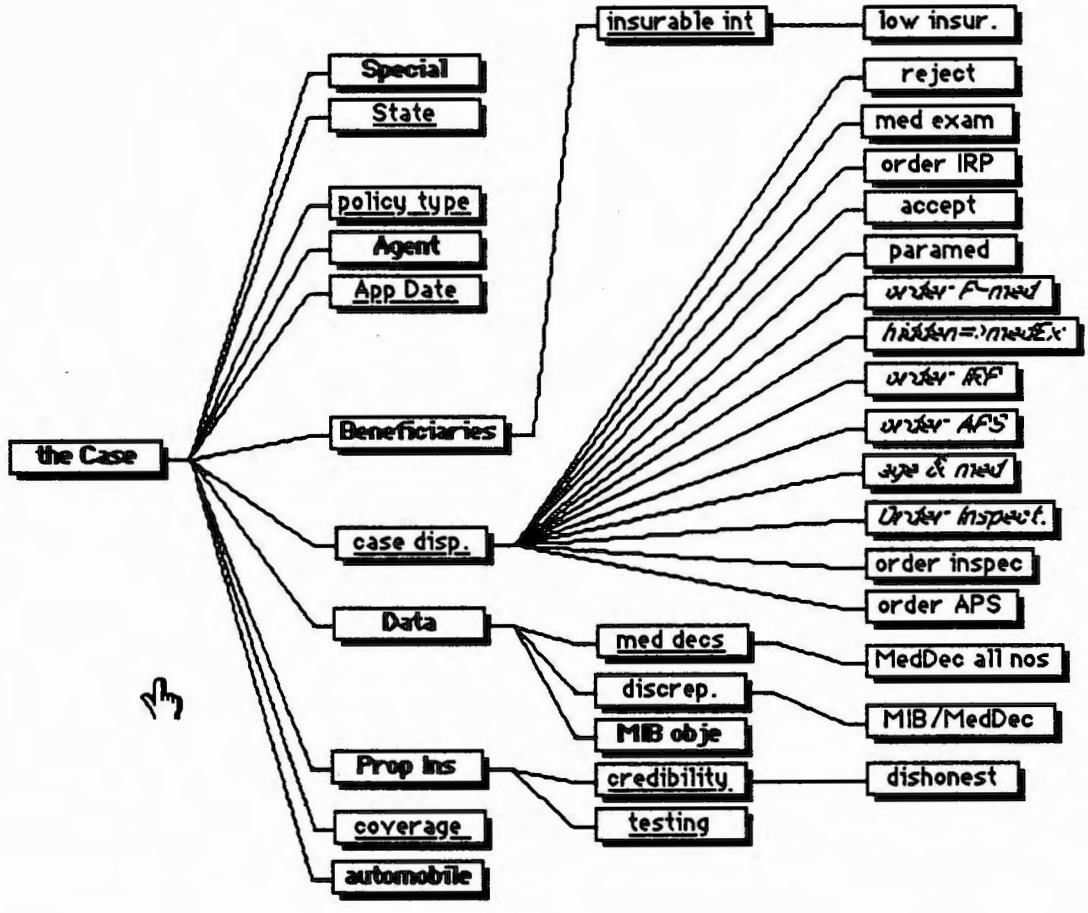


Figure 10. A larger view of the ownership relationships.

CAMEO's "history" facility has also proven useful in group work - especially in later stages of a model's development. When an element is created and each time it is modified (e.g., through renaming, repositioning, or the creation/deletion of links) CAMEO records the event noting the time, date, action and person initiating the action. For any given element of the model, a chronologically ordered list of such events can be viewed by pressing on the "history" button.

DISCUSSION

Current Use

The life insurance underwriting project referenced throughout this article is nearly complete and the resulting model has reached a probable maximum in size at just under 5000 cards in HyperCard distributed among 10 stacks. The model has motivated an update of underwriting guidelines and training manuals. It will soon be used to construct one (and possibly more) expert systems that automate aspects of the life insurance underwriting process.

CAMEO has also recently been used to build:

- a process representation of property and auto insurance underwriting at another insurance company;

- an "investment graph" illustrating the interactions among new retirement fund tax laws;

- the knowledge model and design specification of a new advice component to a widely used software product. This component will assist users on the selection of graph constructs.

Future Work

On the input side we hope to give CAMEO a much more active role in the analysis of transcripts through the inclusion of natural language parser. CAMEO, currently a domain-independent KMS, may also eventually exist in versions that are customized with industry- or company-specific knowledge. Through these efforts the extraction of rule references and the creation of corresponding rule elements in the model may become increasingly automated.

On the output side we plan to complete a routine that can translate CAMEO models into an initial *neural network representation* in one or more popular neural network shells. A good initial representation can greatly reduce the network's time to converge on a solution. Moreover, a model-based initial representation helps to insure that this solution is consistent with human intuitions.

Work is also underway to give CAMEO a better ability to use the frame-based and object-oriented features increasingly standard in expert system shells. New facilities will automate the translation of ownership relationships into code in the same way that CAMEO currently automates the code translation of entailment relationships.

CONCLUSION: THE ESSENCE OF CAMEO

CAMEO is a general purpose, easy-to-use knowledge modeling system with functionality geared towards two basic activities. First, CAMEO helps the user to construct a knowledge model that represents elements and their essential inter-relationships of ownership and entailment. Second, CAMEO provides the tools to use a model, once it has been created, in a variety of ways, to view it from a variety of perspectives, and to gain maximal advantage from its existence.

REFERENCES

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

Boose, J. H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man-Machine Studies*, 23,495-525.

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

Boose, J. H. & Bradshaw, J. M. (1988). Expertise transfer and complex problems: Using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. In J. H. Boose & B. R Gaines (Eds), *Knowledge Acquisition Tools for Expert Systems*, Vol. 2. San Diego: Academic Press.

Buchanan, B. G., Barstow, D., Bechtal, R., Bennet, J., Clancey, W., Kulikowski, C., Mitchell, T.M., & Waterman, D. A. (1983). Constructing an expert system. In F. Hayes-Roth, D. A. Waterman, & D.B. Lenat (Eds.), *Building Expert Systems*, Reading, MA: Addison-Wesley.

Gaines, B. R. & Shaw, M. L. G. (1980). New directions in the analysis and interactive elicitation of personal construct systems. *International Journal of Man-Machine Studies*, 13, 81-116.

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

Shaw, M. L. G. (1981). *Recent Advances in Personal Construct Technology*. London: Academic Press.

Shaw, M. L. G. (1982). PLANET: Some experience in creating an integrated system for repertory grid applications on a microcomputer. *International Journal of Man-Machine Studies*, 17, 345-360.

Shaw, M. L. G. & Gaines, B. R. (1986). Interactive elicitation of knowledge from experts. *Future Computing Systems*, 1, 151-190.

Shaw, M. L. G. & Gaines, B. R. (1988). KITTEN: Knowledge initiation and transfer tools for experts and novices. In J. H. Boose & B. R Gaines (Eds), *Knowledge Acquisition Tools for Expert Systems*, Vol. 2. San Diego: Academic Press.

Wahl, D. (1986). An application of declarative modeling to aircraft fault isolation and diagnosis. Lucker, P. A. & Adelsberger, H. H., Eds. *Intelligent Simulation Environments*. La Jolla, California: Society for Computer Simulation, pp. 25-28.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Graphical Knowledge Acquisition

Robert L. Joseph
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213
joseph@cs.cmu.edu

1. ABSTRACT

Many expert and planning tasks can be represented in a mental model in which an expert manipulates objects from one state to another (delivery route planning - trucks, buildings, packages, routes, etc.; machine shop job scheduling - parts, machine, time, etc; mathematical matrix manipulation - number, rows, columns, etc.). This suggests a highly graphical knowledge acquisition tool where the expert is able to capture the visual intuition of the problem solving in developing the domain knowledge base (KB). By exploring knowledge acquisition (KA) for object manipulation domains, insight will be gained in how knowledge is acquired and represented for such visual-oriented task.

2. INTRODUCTION

KA has been characterized as "*the transfer and transformation of problem-solving expertise from some knowledge source to a program*" (Buchanan *et al.*, 1983, p. 129). In this paper a restricted type of knowledge acquisition will be examined. The paper will focus on the human interactive process involved in building and developing a knowledge base for an expert or planning system where the expert is physically or mentally manipulating objects. By focusing on the acquisition of knowledge for visual planning task, it will be shown how problem solving feedback can be graphically facilitated and improved.

This paper will first discuss the multiple dimensions of KA and what is required to handle them successfully. Then the KA bottleneck problem and what needs to be done about it will be examined. Finally these requirements and needs will be viewed in terms of the KA system APPRENTICE.

3. KNOWLEDGE ACQUISITION

3.1. Dimensions Of Knowledge Acquisition

Knowledge acquisition for a graphical domain is multidimensional. There are different phases of the knowledge base life cycle (create, extend, modify, or maintain), different states of the knowledge base system as errors and gaps in the knowledge are detected (dynamic or static) and different knowledge agents initiating the acquiring of new knowledge or correcting of old knowledge (user or system). These dimensions should be addressed in order to aid the knowledge base development and enhance the knowledge acquisition process.

A graphical depiction of the different KA dimensions is shown in Figure 1 in the form of a block. Each side of the block represents a different aspect of the knowledge acquisition process. Each process will be discussed later in the paper.

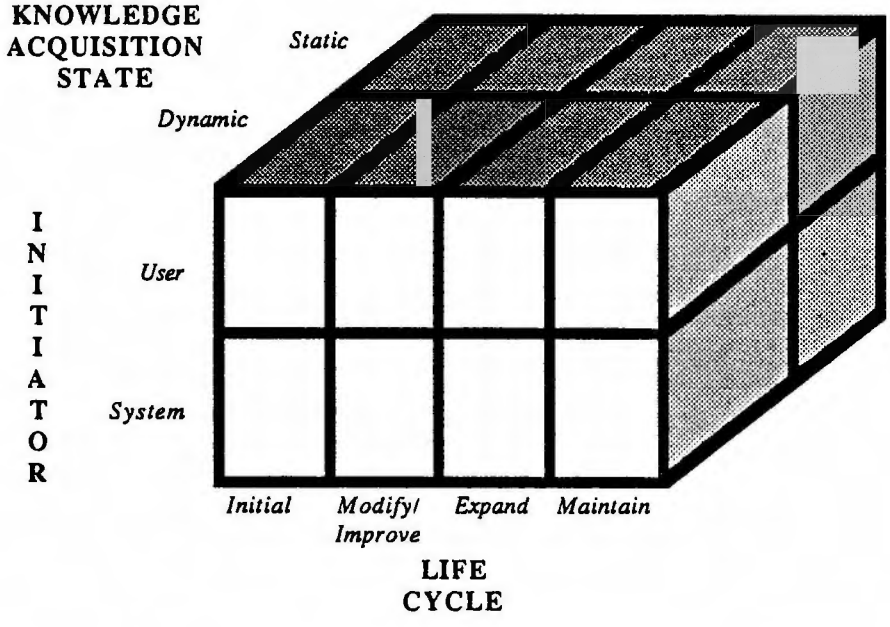


Figure 1: Representation of the Knowledge Acquisition Process

The research being described is domain independent but to help illustrate significant concepts, a domain called "truck delivery" will be used throughout this paper. The trucking domain is as follows:

> A fictitious company called Forest Lines has several buildings around town that are different distances from one another and packages in different rooms in the building. There is a robot in each building that can move packages from one room to another. Each building also has a loading dock. The company owns a truck that can be used to move packages between buildings. All movement takes time (i.e. robots moving packages, trucks moving from one building to another, etc.). The truck can only pick up and deposit packages on the loading dock. The first goal of the domain is to specify a plan that takes an initial configuration of packages and moves them into a final goal configuration. The second goal is to improve the efficiency of the plan (i.e. Move packages to the dock while the truck is in route).

3.1.1. Life Cycle Phases

The most important dimension in KA is the life cycle of a knowledge base. From the time a KB is conceived until it is no longer useful, the KB is being built, developed or maintained. In order to help with the acquisition and maintenance of the knowledge base, the goals for each successful phase need to be understood. It is important to keep in mind that even though these phases are treated separately for expository purposes, during acquisition they are very much interrelated.

A good KA systems should make the phases transparent to the user by providing the right help at the right time. This allows the expert to focus on the information needed by the system. By analyzing and addressing the KA user's needs and goals a better system can be designed.

As stated previously the life cycle phase of knowledge acquisition breaks down into four phases (initial, modify, expand and maintain). The next four sub-sections describe the goals of these phases and some of the requirements that need to be satisfied if these goals are to be attained.

Initial Phase: This phase begins to establish a common ground of communication between the expert and the system. During this phase the expert is:

1) Developing domain concepts (i.e. domain objects and relationships). In the truck delivery domain these are objects such as package, truck, building, etc., and relations such as in-room, on-truck, etc.
2) Developing basic operations that transform one state into another (i.e. domain rules). In the trucking domain this could be load-truck, unload-truck, move-package, move-robot, go-to-next-location, etc.

During this phase the expert needs some framework to aid in the initial definition of the domain. Tools should allow the user to build a domain that is natural for the expert to talk about yet formal enough for the planning system to use. The idea is to create a computational environment which is also intuitive so that the transfer of information form the expert to the computer is straight forward and easy.

Modify/Improve Phase: The modify/improve phase deals with debugging and improving the KB. During this phase the expert needs to determine if the KB system is solving the problem correctly. If the system is not solving the problem correctly, there might be missing or erroneous knowledge that needs to be identified. For example the prerequisites for the load-truck rule might be missing (i.e. a package and the truck have to be at the same building dock).

The expert needs to be able to observe and correct the system as if the system were an apprentice learning how to solve a problem . For example suppose the system has to move two packages from buildingA to buildingB. The system loads package1 on the truck and leaves for buildingB to unload package1. The expert should be able to stop the system and view the choices the system had available to it when it decided to leave package2 behind. If one of these choices was to load package2 on the truck then the expert would tell the system to select the load package2 choice instead of go-to-next-location. The system should then be able to gracefully solicit information from the expert to create a general search control rule to apply during similar decision points.

Expand Phase: Once a KB is initially establish the expert will want to expand the KB to encompass more of the domain. In our domain, the expert may want to add several buildings in different cities and provide the company with access to plane delivery. As the system becomes bigger it becomes more and more important to help the user keep the KB consistent and manageable.

The user needs to be able to view the KB in an intelligent way (Nirenburg *et al.* 1988) This will allow the user to see where new information needs to be added or to check to make sure that the information is not already present in the KB. Also the user should be supported in making sure that problems solvable before the new knowledge is added are still solvable after.

Several systems address the problem of very large knowledge bases. KREME (Abrett & Burstein 1987), ONTOS (Nirenburg *et al.* 1988), and CYC (Lenat *et al.* 1985) are systems being constructed to investigate how to deal with consistency and manageability issues that become so critical when a knowledge base gets very large.

Maintenance Phase: Once the system is being used in a working environment other problems occur. One problem is that changes and corrections must be integrated into a system used by many people. Another problem is that several people might be developing the knowledge base at the same time which creates a version control problem. Many of these issues have been addressed in software development systems (Habermann 1982). During this maintenance process it is most important that the KB is consistent and robust.

3.1.2. The Knowledge-Base System State

What state a knowledge-based system is in during knowledge acquisition depends on whether it is engaged in problem solving or not.

Static System States: A system is in a static state of KA when knowledge is acquired without the system engaging in problem solving. A system is in this state during most of the initial stages of KA when knowledge is first being formulated. While the system is in this state the expert is adding objects, relationships, facts or rules of thumb that are used to solve problems in general. The expert can also put in control knowledge during this stage but generally the expert is supplying knowledge that has a less interactive element to it.

Dynamic System States: A system is in a dynamic state of KA when knowledge is acquired while the system is engaged in problem solving. A system is in this state during most of the modify/improve phase. While in this state the expert determines whether the system is solving problems correctly and if not, identifies the erroneous or incomplete knowledge. During problem solving, the system is constantly in a different state because the elements of the system are interacting. The expert is therefore involved in encoding knowledge about the interworkings of knowledge elements. This state is similar to the state a system is in during the debug and test cycle (Buchanan *et al.* 1983).

3.1.3. Initiator Of Acquisition

An inconsistency or error in the knowledge base can be detected either by a computer system automatically or by the expert based on the system's behavior. Because a computer system can process things faster and more accurately, it would be preferable if the system could always detect problems with the knowledge base and then inform the expert. But with current technology, this is not completely possible. It is possible however for a system to provide limited help in flagging potential KB problems. This help can be initiated automatically by the system in the form of heuristics to detect problems or by request from the user when a problem is suspected. For example the user can ask the system to check possible incorrect knowledge base information such as whether there are any objects defined that are not used anywhere. On the other hand the system can flag possible knowledge base errors without user prompting such as informing the user of predicates that were not used by the system during problem solving. By investigating what the system can do to help initiate knowledge acquisition or what tools can help aid in the quick identification of KB problems a more effective system is produced.

3.2. KA as a Bottleneck

There are several reasons why KA is a bottleneck in building expert and planning systems. The major difficulty is finding a common ground of communication between the expert and the system. Reaching common ground is also tied to the problem of determining how to organize the domain knowledge in the system. Even when these problems are solved there is still the problem of encoding a lot of domain knowledge into the system to provide expertise. Finally the process for soliciting this domain knowledge is still not clearly understood.

3.2.1. Need for expert to directly input knowledge

Knowledge-based systems are frequently built by a domain expert working with a knowledge engineer. The domain expert describes the domain and problem solution to the knowledge engineer and the knowledge engineer encodes the information into the system. The building of the knowledge base is a cyclic, three-phase refinement process: 1) The background information about the domain is outlined; 2) The problem and expert solution is described and; 3) The systems knowledge base is debugged as the system solution is compared with the expected expert solution. Unfortunately problems arise because the domain expert does not fully understand the interaction of the system and the knowledge engineer does not fully understand the domain knowledge. As a result knowledge acquisition process becomes both time consuming and error prone.

- The domain expert may incorrectly explain a process because of insufficient understanding of the meaning of various elements of the system or
- The knowledge engineer might incorrectly translate the domain expert's explanation into system elements because of inadequate understanding of that explanation.

To avoid this dilemma the system should allow the domain expert to create, run and debug the knowledge base keeping the system architecture invisible to the expert.

3.2.2. Debug-Refine-Modify cycle is a huge task

The debug-refine-modify cycle is a huge task. Expertise requires large amounts of information which has to be encoded in an expert system. Often there is a complicated interaction among the facts that an expert uses to solve a problem. The expert may not be able to explain this interaction on a conscious level, therefore making the problem of acquiring the knowledge more difficult. Finally the system domain knowledge representation may not fully agree with the expert's. If the difference is great enough the system's representation may have to be changed to conform better with the expert's.

3.2.3. Need for understanding the entire KA process

Buchanan and others (Buchanan *et al.* 1983) think of the KA process as Identification, Conceptualization, Formalization, Implementation, Testing, and Revision. This defines the knowledge base development life cycle but does not formalize the other dimensions of the KA process. Further the Buchanan definition deals more with the initial knowledge acquisition process in a higher grain definition than the one described here. Below is a conceptual mapping between the two definitions.

Present paper	Buchanan *et al.*
(not addressed)	Identification
Initial phase	Conceptualization, Formalization, Implementation
Modify/Improve	Implementation, Testing, Revision
Expand	(partially addressed)
Maintain	(not really addressed)

The research on ontological analysis is being used for the preliminary analysis of a problem-solving domain (Alexander *et al.* 1987). This technique defines static and dynamic ontological groups as well as an epistemic group. This breaks knowledge base information into functional solving parts. However this particular technique of ontological analysis focuses only on the initial stage of KA.

Interviewing is another way to acquire knowledge from experts, but a knowledge engineer still has to translate the information into the system. Thus the problems previously outlined still exists.

4. APPRENTICE - THE KA SYSTEM

4.1. Introduction

When an expert teaches a student apprentice a new skill, the expert starts off by giving the student facts about objects in the domain including facts about attributes, relations and behaviors. By doing this the expert is building up a common vocabulary between the expert and the student. After a small amount of knowledge is given to the student, a test problem is provided. The expert watches closely as the student attempts, step by step, to solve the problem. If the student makes a mistake, the expert sets about determining why the student made an error. Was it because the student lacked some important information? Or was it that the student misinterpreted the expert's information? The process of teaching the student about the domain can be viewed as a static knowledge gaining process, whereas correcting the student solving a problem can be viewed as a dynamic knowledge gaining process. The design of APPRENTICE is based on this apprenticeship paradigm and pays close attention to the static/dynamic distinction.

The system design also addresses the different phases a knowledge base goes through. Understanding these phases is important to help:

1) characterize needed system tools,
2) provide some structure and order to a very complex problem,
3) allow knowledge to be analyzed more completely and
4) provide a framework to obtained valid knowledge.

Static KA generally needs tools for developing a working vocabulary about the domain. The formation of objects, relationships and operators for a domain are the main concerns of this process. Because there is much similarity between domain objects (e.g. loading-dock is a special type of room) information about a "parent" object can be used to aid in the definition of a "child" object. This allows information about the parent to be inherited or used by the child. Being able to define objects in a hierarchical fashion reduces coding and makes the knowledge base more consistent. Inheritance may also be useful in the definition of relations, operators and state definitions. This is a research area that will be investigated in the future.

On the other hand the dynamic process generally needs tools for developing, viewing and debugging a knowledge base when the system is trying to solve a problem. By allowing the expert to inspect the solution steps of a task the expert is able to understand more fully what the system is doing. Providing tools and techniques that will help with this inspection will aid the expert in determining what knowledge is missing or incomplete in the knowledge base.

Although static and dynamic KA seem to be distinct, it may take a combined interaction between these different types of KA to solicit the needed knowledge. Also an ideal KA system would initiate all knowledge acquisition opportunities so that all the expert would have to do is answer questions. With current technology, this is not yet possible, but it is possible for the system to sometimes intervene just as a student apprentice sometimes intervenes by asking questions of the expert (Kahn *et al.* 1987). When and how such a mixed-initiative or interactive KA system can do this is currently being investigated.

4.2. System Overview

APPRENTICE is being developed to experiment with multi-dimensional knowledge acquisition. It is being used to develop knowledge bases use by the planning system PRODIGY[1] (Minton *et al.* 1989). The target environments for the system are the Macintosh running Allegro Common Lisp

[1] PRODIGY is a general purpose problem solving architecture that serves as the basis for research in planning, machine learning, apprentice-type knowledge-refinement interface and expert systems.

and the IBM-PC RT running CMU-Lisp and X-windows. In order to support these two environments a machine independent graphical package, FrameGraphics (Joseph, 1989), was developed[1]. A top level view of the system architecture is shown below.

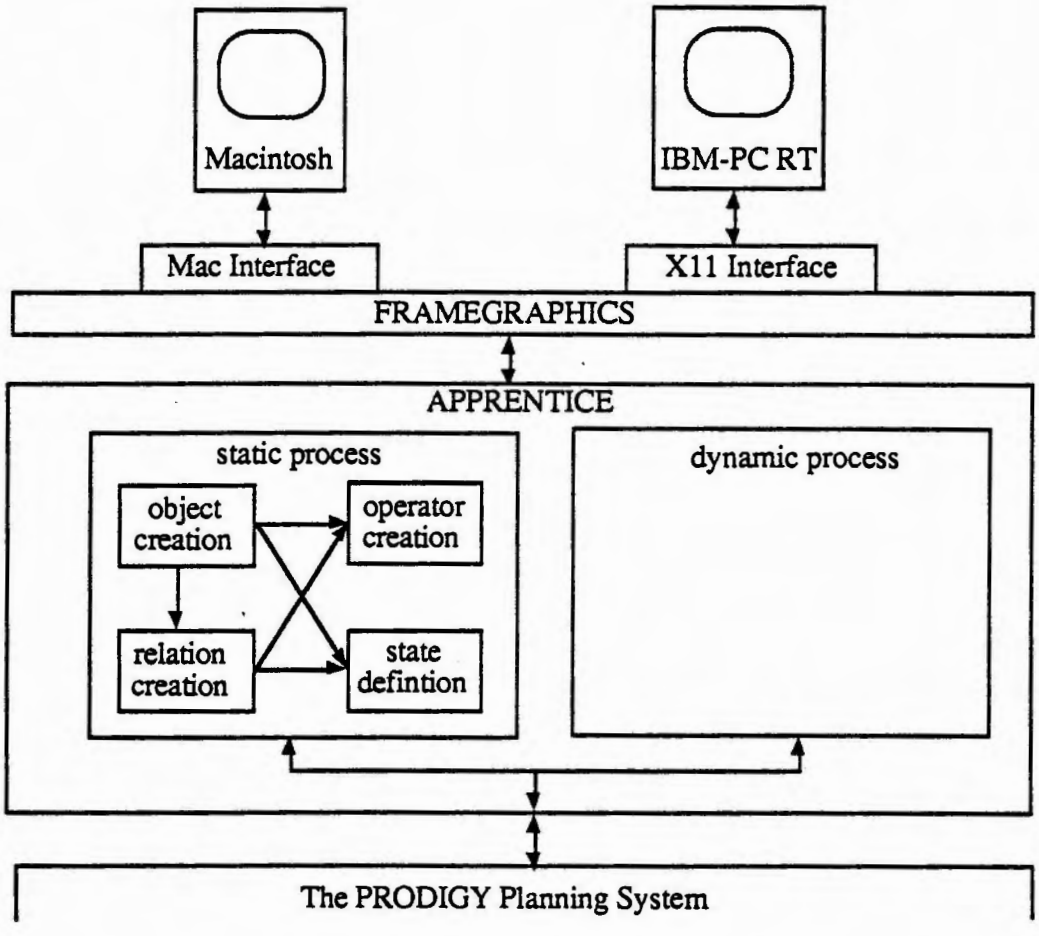


Figure 2: Architecture Overview of APPRENTICE

During the static phase objects created are used to create relations. Objects and relations created are used to define operators and states. The static process defines a domain to be run in PRODIGY. These creation subcomponents will be outlined in the following sections. A small portion of the truck delivery domain (moving a robot from room to room) will be used to illustrate each subcomponent.

[1] FrameGraphics is a semi object-oriented graphical tool using FRAMEKIT (Nyberg, 1988) as its representation language.

4.3. System Description

4.3.1. The Static System

General Window Description: APPRENTICE's static system has five windows. The four subcomponent windows (object, relation, operator, and state) are used to define a domain. The apprentice window is used to create a problem that will be run in PRODIGY. All the windows have some similar parts. This similarity allows the user to interact with each window in a consistent manner.

All the windows have a rectangular box in which the window's components are defined. For example in the object window's rectangle the object room-model can be defined. Only one component can be actively defined at a time but other components (partially or fully defined) can be present as icons. With a quick double click of the mouse the inactive icons can become the active component while the active component becomes an icon.

Each window has text items in it that are either system information or domain definitions. System information is used by the system to communicate with the user. This information can not be changed. Domain definitions are information need to define a domain. This information is set to default values initially but the user can edit these values very easily. This editing ability is consistent in all the windows.

Finally each window can be customized to each user's own preferences. Each item on the screen can be easily moved wherever the user desires. This allows the user to personalize the environment to obtain the most effective layout.

To represent state in prodigy and to represent operator pre-conditions PRODIGY uses predicates. A predicates is represented as a list - a predicate name followed by 0 or more elements. These elements can be variables or instances. The model and operator windows are used to create the predicate skeletons and the operator and state windows use these predicates to further define the domain.

Object Creation Window: The model-window is used to define the objects in a domain. The objects are defined one at a time both graphically and semantically. The graphical definition consists of a graphical representation of the object along with specific regions designated as hotspots. These hotspot regions are points, lines and areas in which other object hotspots can be connected[1]. The semantic part of an object consists of an attribute list, the predicate representation, an instance variable name and the parent object specification. The user can build a complete library of objects for a domain in the model-window. The figure below depicts the model window in the process of creating the object room-model.

[1] Currently the graphical representations are created with a text editor. It will be simple to add a graphical editor for the simple line drawings that are done.

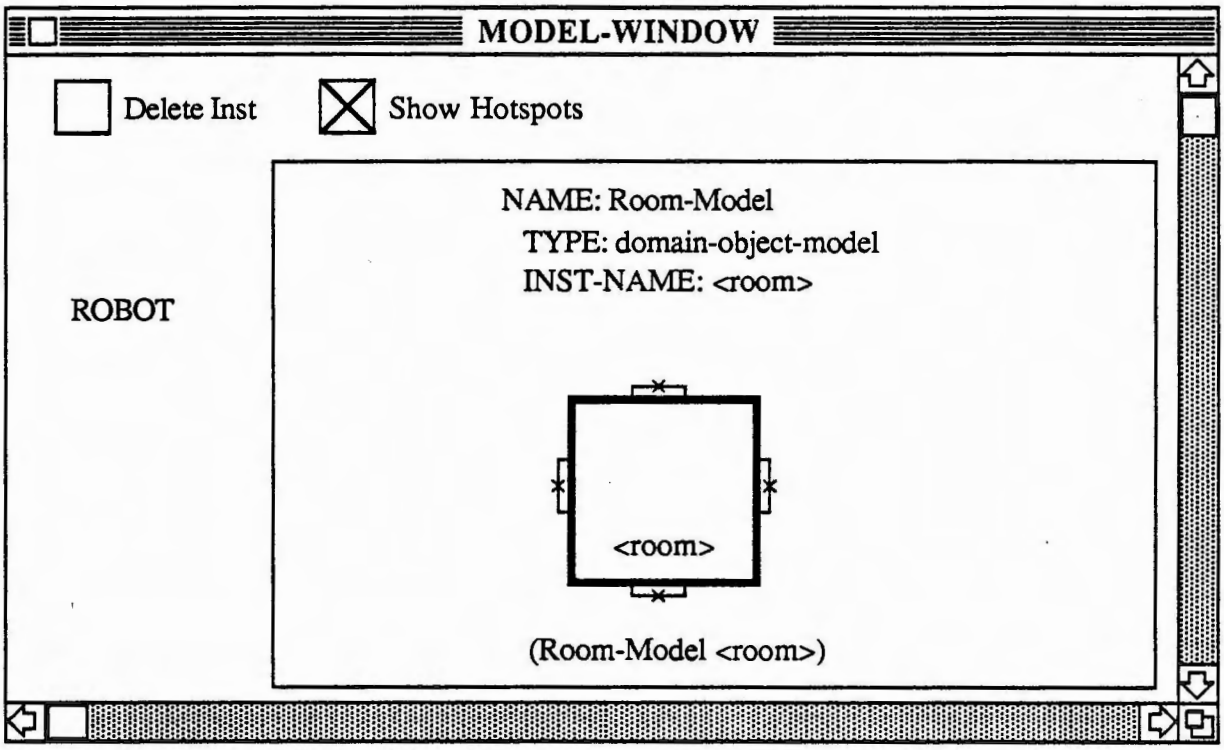


Figure 3: The object Room-Model is being defined

The graphical depiction of a room is shown along with five hotspots. The x on each side of the room defines four hot points that designate the doorways for a room. This will allow other objects to be connected to the doorways (i.e. other rooms). The hot area inside the room is designated by the thick lined rectangle. This hot area allows objects to be be placed inside a room (i.e. robots and packages). The hotspots can be displayed or not allowing the user a less cluttered view.

The object Room-Model is defined to be a type DOMAIN-OBJECT-MODEL. This is the highest parent object that the system knows about. If we were defining a loading dock the type could be Room-Model. Currently Room-Model has no attributes. However if the color of a room where important, for example, we could have the attribute color for Room-Model. The predicate skeleton "(Room-Model <room>)" is used by the PRODIGY system to define an object in a domain. This is presently generated automatically by the system but can be edited by an expert user. Notice that the robot icon is to the side. This contains the definition of the robot object and can be chosen to be worked on at anytime.

Relation Creation Window: The relation window allows the expert to build relations between objects. These relations or connections between objects are defined so that they can be automatically recognized and generated in the operator and state definition phases. By allowing this automation, experts will be able to define much of the rule and state knowledge graphically and the system will be able to check and flag inconsistencies in these definitions. Below is an example view of the relation window.

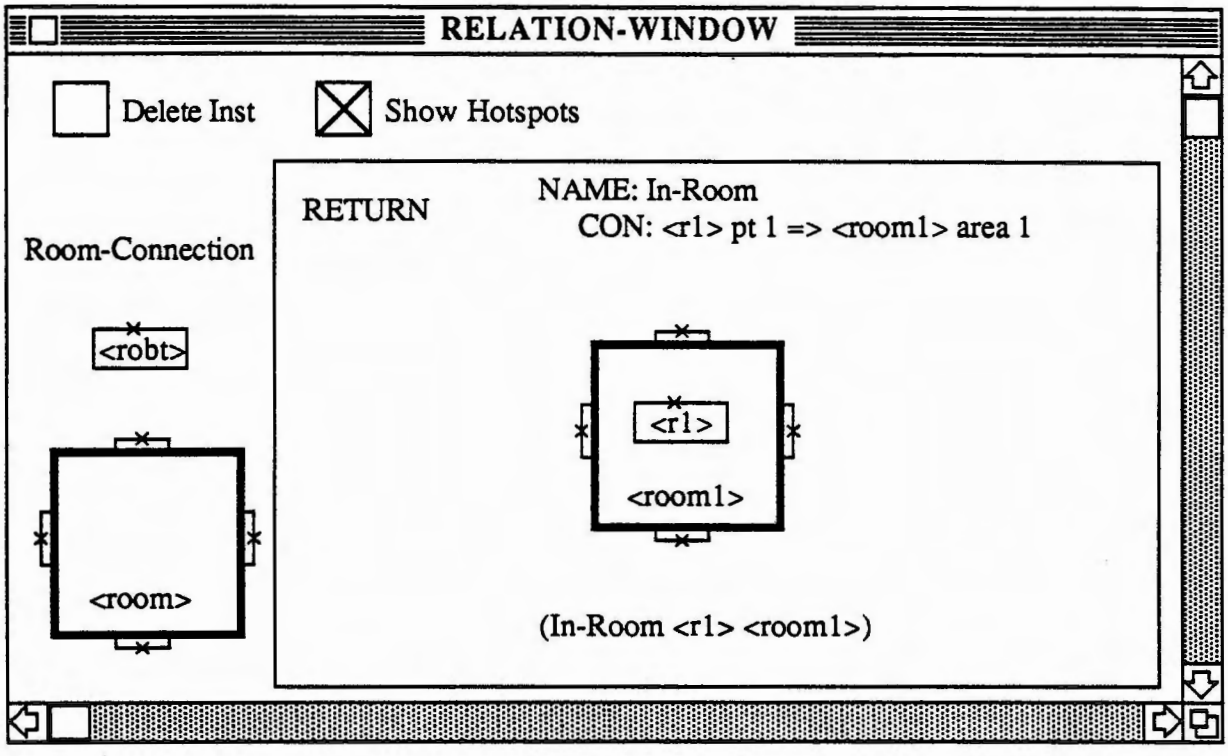


Figure 4: The relation In-Room is being defined

In the Relation-Window object prototypes that the system knows about are to the left of the relation definition box. Relations are defined by dragging a prototype into the rectangular area which produces an instance of the object. This instance is now part of the current relation definition. In the example, In-Room is the relation being worked on while the relation Room-Connection is inactive.

The definition of the relation in-room has <robot>'s hot point inside the <room>'s hot area. This hotspot connection is made explicitly by the user. The <> designate that the objects in the definition are variables. The textual item (In-Room <r1> <room>) is a predicate definition that corresponds to a robot in a room. Thus if a state were defined with a robot called NOMAD, a room called the Bridge and the robot was in the room area then one predicate in the state definition that could be automatically generated would be (In-Room NOMAD Bridge).

Operator Creation Window: The operator window allows the creation of operators or rules in the system. The operator is initially defined by building a picture of the precondition state needed before the operator can fire and then building the post condition state after the operator has fired. Building an operator state is analogous to building a relations state. This allows the expert to visually define how an operator works while also leaving much of the bookkeeping work of the state changes for the system to compute. There may still be some information that has to be added to the operator definition because all the information may not be captured by the visual state but much of the information is available. Below the operator move-robot is being defined.

The preconditions for move-robot are defined by the state that is being depicted in the pre section of the window. Since variables are used in this definition the operator is general. If any room and robot matches these preconditions then the operator can fire. Predicates will then be added or deleted from the state to reflect this effect. The pre-state and effect information has been completely generated automatically from the above graphical pictures along with the previous

object and relation definitions. There also may be additional information that is not apparent in the state but may need to be added by the expert. For example, if we were dealing with time for the robot, then in the precondition we might have a predicate (Time <r1> <x>). Then we would have to add the following two clauses in the post condition: (del (Time <r1> <x>)) and (add (Time <r1> (1+ <x>)))[1].

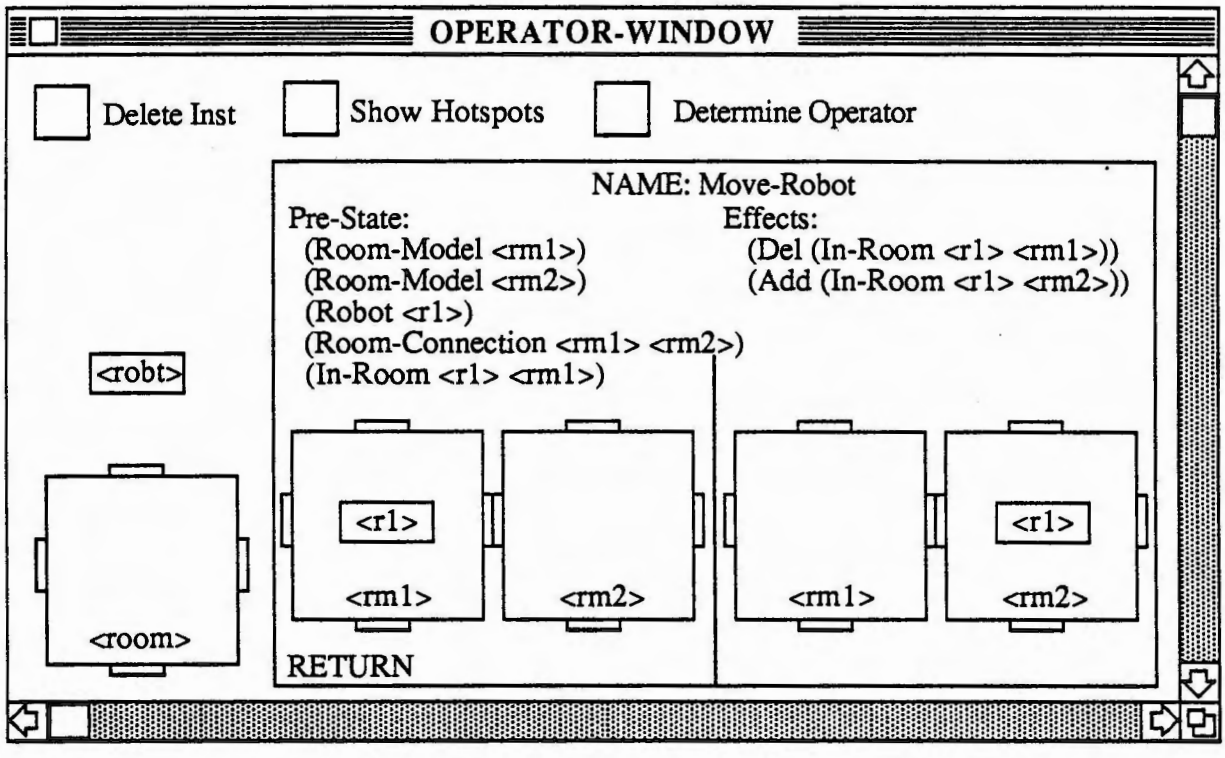


Figure 5: The operator Move-Robot is being defined

State Description Window: There are two states that are important in defining a problem for a system, initial and goal. The initial state is a statement of the entire world before the system is run. This initial state can be defined by having instances of objects connected together. With this visual representation the system could generate an initial state definition automatically. Figure 6 below is a visual depiction of three rooms with an automatically generated state description. Notice that now the rooms and robot have names associated with them making them instances of their parent objects.

The goal state is a little different from the initial state because the goal state can be partially defined. One can have a goal that does not use all the predicates of a domain. It could also use variable objects. For instance one may want to have a goal of having the robot robby in any room next to RoomC. The graphical picture in figure 7 below shows this representation.

[1] This is not quite how to define the post conditions but it serves to illustrate the point.

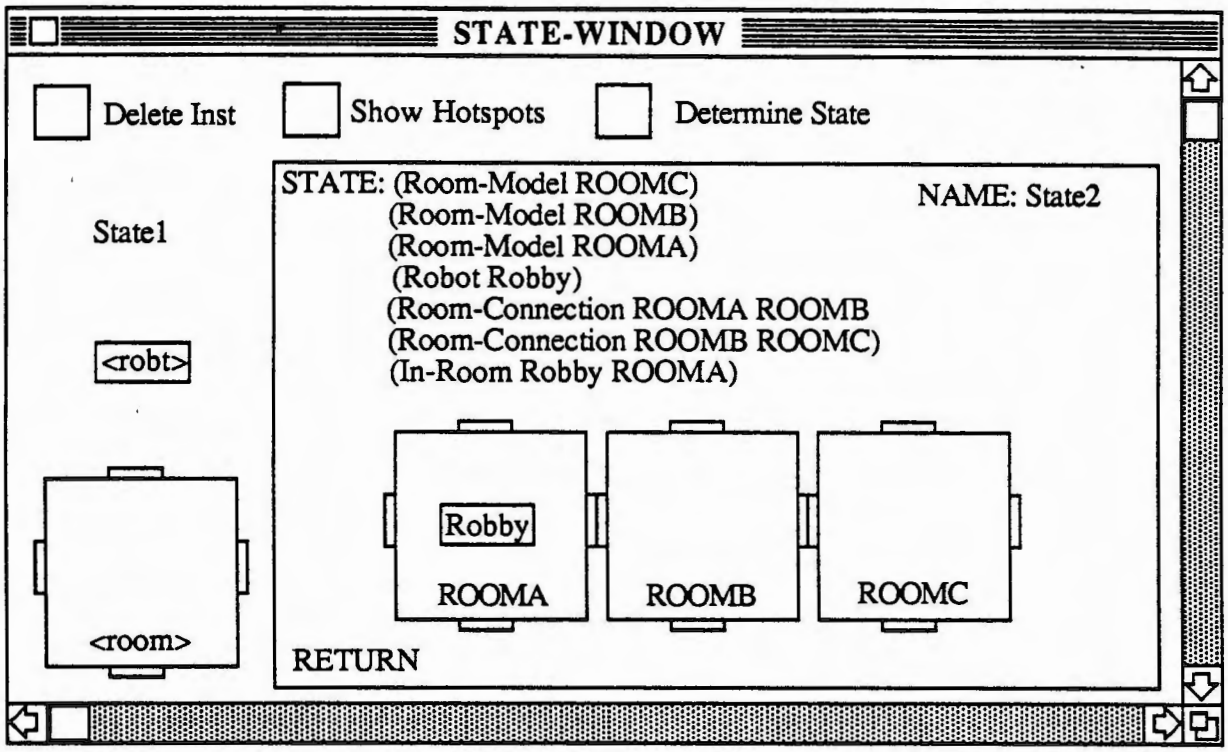


Figure 6: An initial state definition

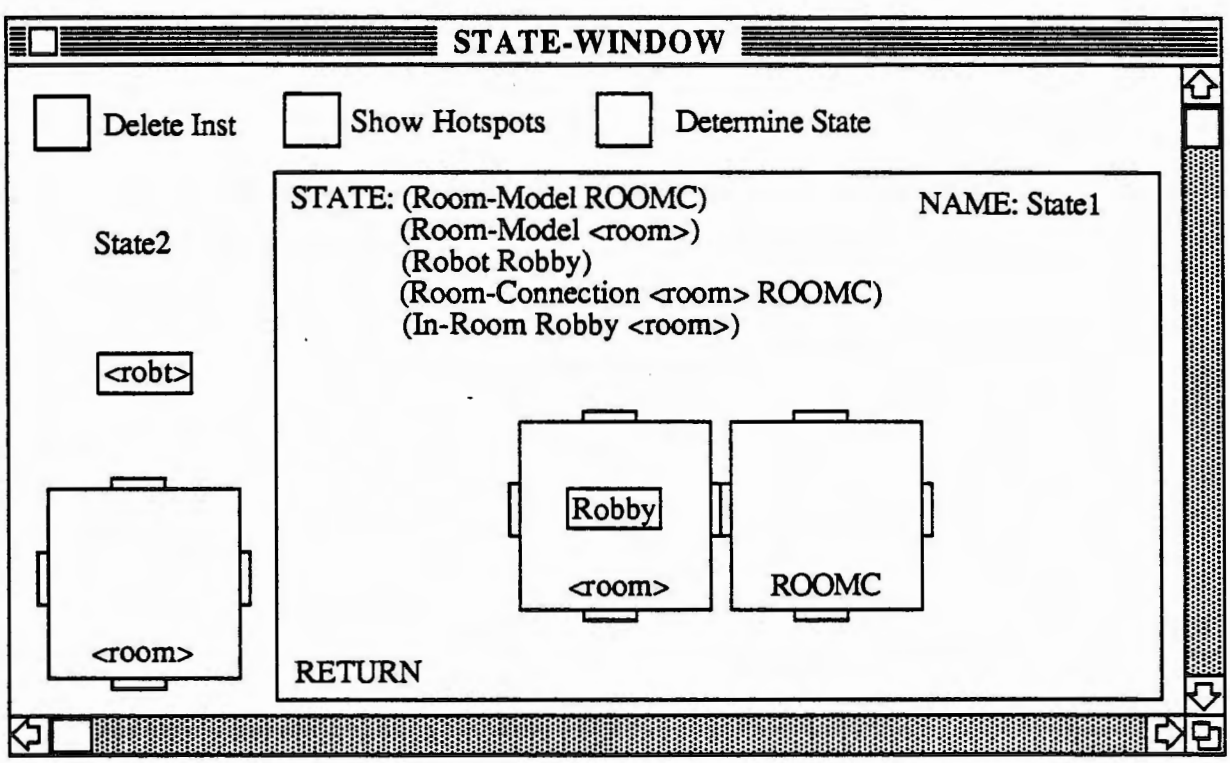


Figure 7: A goal state definition

When defining a goal state the user will have the ability to edit the state definition. As with the operator this is useful when information is not captured by the graphics or it is preferred that the state be under specified.

Running The System Window: In order to solve a problem prodigy needs several things: 1) the operators that are being used, 2) the starting state for the problem and 3) the goal state for the problem. These are specified separately because different parts of the problem may be interchanged. For example one may be solving several problems with the same initial state but different goal states, or a user may want to test out only a subset of the available operators. Below is the APPRENTICE-WINDOW that allows the user to define problem. By selecting the Run Prob button PRODIGY is run with the active problem.

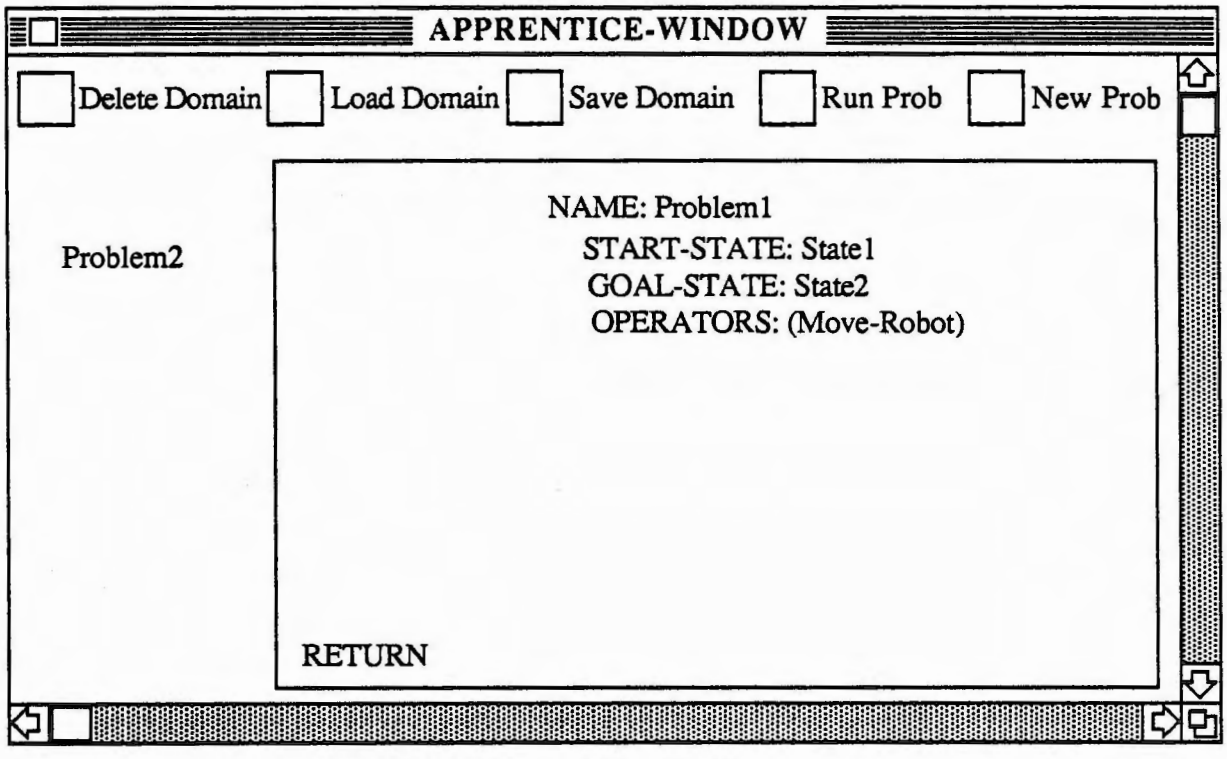


Figure 8: Test problem being defined

4.3.2. Acquired Knowledge

In the above examples the system acquired the knowledge necessary to move a robot from one room to another. The object creation phase enabled the expert to define the objects robot and room. The relation creation phase enabled the relation in-room to be defined. The operator phase created the operator move-robot and finally the state creation phase enabled the expert to create a problem with an initial and goal state to have the system solve. The expert can continue to add more knowledge to the domain, incrementally building a more complete knowledge base. For example the expert could very easily add a package object and allow robots to move packages from room to room. Below is an actual trace of the system solving the above problem.

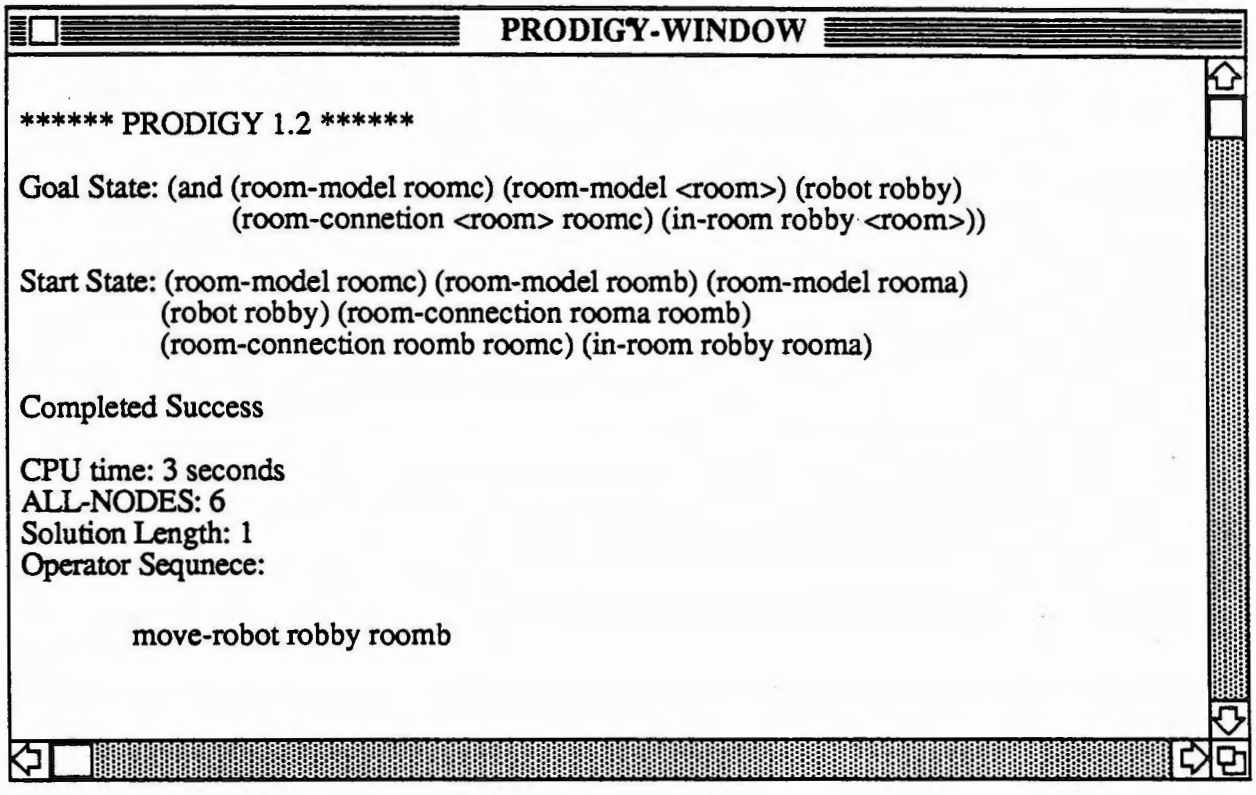


Figure 9: Trace of PRODIGY running

4.3.3. The Dynamic System

Much of this research is still to be done. In the next stage of development, the system will allow problem solving graphically in a step by step fashion. Once the initial state has been graphically defined the system can graphically reflect the application of operators allowing the expert a more visual picture of how the system is solving a problem. The expert can then interrupt the solution process when things do not look correct. This is similar to how an expert teaches an apprentice.

5. CONCLUSION

Knowledge acquisition for a task requiring physical or mental manipulation of objects can be improved by understanding the KA process and providing graphical tools for developing domain knowledge. This research characterizes the multiple dimensions of the KA process and outlines a system in which graphical tools are smoothly integrated into representational techniques that are responsive to these multi-dimensional KA needs.

I would like to give my sincere thanks to my advisor Jaime Carbonell. His guidance and understanding has been invaluable to me. Thanks to Gary Kahn and Dario Guise who have helped me focus my views. Finally many thanks to Ira Monarch for his many suggestions.

6. REFERENCES

Abrett, Gleen & Burstein, Mark H. (1987). The KREME knowledge editing environment. *Knowledge Acquisition Tools for Expert Systems*. Academic Press Inc; San Diego CA 12101, pp 1 - 24.

Alexander, J.H., Freiling, M. J., Shulman, S.J., Rehfuss, S. & Messick, S. L. (1987). Ontological analysis: an ongoing experiment. *Knowledge Acquisition Tools for Expert Systems*. Academic Press Inc; San Diego CA 12101, pp 25 - 38.

Boose, John H. & Bradshaw, Jeffery M. (1987). Expertise transfer and complex problems: using AQUINAS as a knowledge-acquisition workbench for knowledge-based systems. *Knowledge Acquisition Tools for Expert Systems*. Academic Press Inc; San Diego CA 12101, 1988, pp 39 - 64.

Buchanan, G. G., Barstow, D., Bechtal, R. Bennett, J., Clancey, W., Kulikowski, C., Mitchell, T., and Waterman, D.A. (1983). Construction of an expert system. In Hayes-Roth, F., Waterman, D.A., and Lenat, D. . (editors), *Building Expert Systems*, pp 127-167. Addison-Wesley, Reading, Massachusetts.

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules. *Artificial Intelligence*, **12**, pp 121 - 158.

Habermann, A. Nico and Notkin, David S. (1982) The Gandalf Software Development Environment. In *The Second Compendium of Gandalf Documentation*, Carnegie-Mellon University.

Joseph, Robert L. (1989). FrameGraphics 0.5 Manual, Computer Science Department, Carnegie-Mellon University (Unpublished).

Kahn, G. S., Breaux, E. H., Joseph, R. L. and DeKlerk, P. (1987). An intelligent mixed-initiative workbench for knowledge acquisition. *Knowledge Acquisition Tools for Expert Systems*. Academic Press Inc; San Diego CA 12101, pp 161 - 174

Lenat, D., Prakash, M., and Shepherd, M. (1985). CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. *AI Magazine*, **6**, pp 65 - 85.

Minton, Steve, Knoblock, Craig A., Kuokka, Daniel R., Gil, Yolanda, Joseph, Robert L. & Carbonell, Jaime G. (1989). Prodigy 2.0: The Manual and Tutorial, Computer Science Department, Carnegie-Mellon University.

Musen, Mark Alan (1988). Generation of Model-Based Knowledge-Acquisition Tools for Clinical-Trail Advice Systems, *Doctoral dissertation*, Computer Science Department, Stanford University.

Newell, Alan, (1981). The knowledge level. *AI Magazine*, **2**, pp 1 - 20.

Nirenburg, S., Monarch, I. Kaufmann, T. and Carbonell J. (November 1988). "Acquisition of Very Large Knowledge Bases: Methodology, Tools and Applications", *Third AAAI-sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada

Nyberg, Eric H. (1988). The FRAMEKIT User's Guide Version 2.0, Computer Science Department, Carnegie-Mellon University.

Pace, Luigi Di & Fabrocini, Filippo (1988). Using Classification to Guide Knowledge Acquisition and Refinement in Real-World Domains, (submitted to IJCAI-89).

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989

Analogy In Knowledge Acquisition: Description of An Analogical Tool

Debbie Leishman - Knowledge Science Institute, University of Calgary
& Institute for Research in Construction
(leishman@cpsc.UCalgary.CA)

Abstract

Learning and reasoning by analogy provides one mechanism for acquiring and effectively using knowledge. This paper describes the development of an Analogical Tool that performs this learning and reasoning and which is based on evaluations of partial correspondences over conceptual graphs. The tool results from instantiation of a general framework for analogy that arises from analysis of previous work and characterizes analogies as "common generalizations". Using conceptual graphs [Sowa, 1984] as the knowledge representation scheme for the general framework, the tool forms analogies and performs analogical inferencing. Analogy formation in the tool derives minimal common generalizations as preferred "stronger" analogies. This formation is under the control of four formation evaluations and two ordering evaluations. The paper then presents and example of learning problem solving schemas and their subsequent use in solution hypothesis.

1 Introduction

Analogy has many possible uses in a knowledge acquisition cycle. In the early stages of elicitation, experts often prefer and find it easier to input their knowledge in the form of cases or descriptions of problem solving situations rather than in explicit rules [Witten & MacDonald, 1987]. Analogy can be used here for taking these cases and forming general problem solving schemas which can later be used for reasoning about a similar case. Descriptions of these cases may come from text as well as from an expert and may require complex descriptions. The Analogical Tool described in this paper performs this kind of schema induction and subsequent solution hypothesis. Analogy can also be used to determine similarities and differences between input of multiple experts to a knowledge acquisition tool. Further on in a knowledge acquisition cycle, analogy can be used for exploration in an existing knowledge base. Analogies found between structures can provide a basis for hypothesizing new knowledge followed by evaluation of the plausibility of this knowledge. Similarly, in a knowledge base of cases, analogy can be used to determine where and how to incorporate new cases. Analogy is also useful in the very simple case of copy and edit when adding new facts to a knowledge base as in the CYC project [Lenat, 1986].

The Analogical Tool described in this paper represents complex analogues taken from natural language and forms minimal common generalizations as preferred analogies. This tool could be used for any of the purposes described above but will be described here as forming problem solving schemas and later using them for solving similar problems.

This paper will first develop a general framework for analogy which instantiates to the Analogical Tool when conceptual graphs are used for the knowledge representation scheme. Following this, the Analogical Tool will be described in terms of forming analogies and perfoming analogical inferencing based on these analogies. Finally the tool will be used on an example from Gick and Holyoak to produce schema abstractions.

1.1 Analysis of Previous Work on Analogy

Analysis of colloquial connotations and psychological processes of analogy has generated research which can be classified as 1) Abstract theories of analogy; 2) Psychological models of analogy; and 3) Evaluation criteria for analogy.

Abstract theories view analogy as an algebraic or logical process. Comparison of works in this area reveals the common theme that analogy is the formation of a plausible partial correspondence between parts of the participating analogues. Indurkhya [1985], for example, describes a theory of analogy and metaphor called Approximate Semantic Transference which forms AT-MAPS. These maps represent an approximately coherent partial mapping of terms between source and target domains. Gaines and Shaw [1982] explicate analogy as a partial correspondence between systems represented as mathematical categories. Russell's use of determinations [Russell, 1987] also requires the formation of a plausible partial correspondence between terms.

Psychological models of analogy attempt to account for how and in what circumstances people reason from analogies. Comparison of works in this area reveals the common theme that a critical part of any model of analogy is the knowledge that is used and how it is represented. For instance, Gentner's Structural Mapping Theory [1983] is based on a principle of systematicity which attempts to maintain chains of relations when forming correspondences between domains. The choice of higher order predicate calculus as the knowledge representation scheme aids in detection of these chains.

Evaluation criteria for analogy focus on the problem of what is a plausible analogy—why one analogy is seen as more plausible than another. Comparison of works in this area reveals the need to control the formation of plausible partial correspondences through evaluation criteria which relate to psychological notions of plausibility and the applicability of analogies. For example, evaluation criteria which use generalization/ specialization hierarchies are central to the CYC project being conducted at MCC [Lenat et al, 1986], as well as to the work of Hofstadter and Mitchell on analogical reasoning [1988]. Winston's [1981] early paper on analogical reasoning contains many useful criteria such as the use of case relations and matching of terms sharing the same relations between them; this latter criterion is also evident in Gentner's work.

1.2 A General Framework for Analogy

The above discussion based on classifying works on analogy into the three categories leads to conclusions which can be seen as comprising critical functions of analogy. Any computer emulation must thus contain the following three functions:

1) Semantic schemas for representing knowledge of the domains involved in the analogies.

2) Algorithms for finding partial correspondences between abstract structures.

3) An Evaluation relation for ordering analogies by plausibility.

These critical functions can also be seen as arising from the description of analogy given by Polya [1954]. He describes analogy as a type of plausibility reasoning showing similarity on a definite conceptual level and defines analogy in terms of forming a "common generalization" of analogues.

From this description by Polya we can describe a general framework for analogy which contains within it the three critical functions. We still need some kind of semantic schemas for representing knowledge of our domains, but we can now view the formation of partial correspondences between analogues as arising naturally from formation of a common generalization of the analogues. Those parts of the analogues which have been generalized to a common concept are also those which are put into correspondence. Also implicit in Polya's description of forming a common generalization is that there is often more than one possible common generalization which can be formed. In an effort to find these "stronger" common generalizations we need some kind of ordering relation. This points to our third critical function, the need for an evaluation relation for ordering analogies by plausibility.

The general framework for analogy in Figure 1 shows two analogues and their common generalizations. Each analogue consists of parts, and each of these parts is "part-of" the analogue. The common generalization also contains parts and these parts are each a "generalization" of some part of each of the analogues. The parts of the two analogues which have a common generalization define the parts which are in correspondence with each other. Finally we need an ordering relation which determines the "stronger" analogies. In most applications of analogy, the "stronger" analogies correspond to minimal common generalizations. Figure 1 shows there can be many common generalizations formed but only some are minimal. These generalizations are minimal within our ordering relation because they contain the most information; they are the most specific of the possible generalizations. There can typically be more than one of these minimal common generalizations.

Analogical inferencing can also be described in the terms used above. To perform inferencing analogically from a minimal common generalization means to look for an even more specific generalization. Usually one analogue (the source analogue) is better known than the other (the target analogue). The source analogue is often used to suggest the existence of non-evident structure in the target analogue. In this way some "part-of" the source analogue is hypothesized as existing in the target analogue.

2 Conceptual Graphs for Analogy

Use of conceptual graphs [Sowa, 1984] as the knowledge representation scheme for the general framework described above leads to development of an Analogical Tool. A single conceptual graph states a proposition and contains concept nodes, relation nodes, and arcs between them. This structure allows analogues to be represented as finite bipartite graphs. Concept nodes represent entities, attributes, states and events while relation nodes specify relationships between concepts and include case relations among others. Each concept in a conceptual graph is part of a lattice of concepts. This lattice supports inheritance, and concepts higher in the lattice are more general than lower ones. Concepts to be used in a domain must be specified beforehand.

Graphs can also express more complex sentences that contain individuals, modality, nested propositions and co-references as shown in Figure 2. The sentence for this graph is shown at the top. Co-reference links are shown with dashed lines and the nested propositions are shown by the boxes surrounding subgraphs. Concepts in a conceptual graph can be generic, such as [Person] or can refer to particular individuals such as [Person: Sam]. This concept refers to a specific individual *Sam* of type *Person* where *Sam* is specified in the referent field of the concept. The modal concepts *Think* and *Believe* in Figure 2 take nested conceptual graphs as their objects. These nested graphs are *Propositions* whose referent field is the set of conceptual graphs being asserted. They are also referred to as context boxes and a box with no type label defaults to a proposition (context) box. Also present in this figure are co-reference links which show anaphoric references and express an equality relation between concepts. Co-reference links often join pronouns to their referents or refer to a concept previously mentioned. In Figure 2 for example the phrase, "Sam thinks that the house has a kitchen," refers to a house that Sam already knows about. This requires a co-referent to house outside the context of the house having a kitchen.

Each conceptual graph is also embedded in a larger semantic net which contains background knowledge that enriches the meaning of the graph.

2.1 Specialization and Generalization

The conceptual graph model defines specialization operations used for selectional constraints on sentence formation and generalization operations for logic and model theory. Specialization of a conceptual graph can be done in two ways. First a particular concept

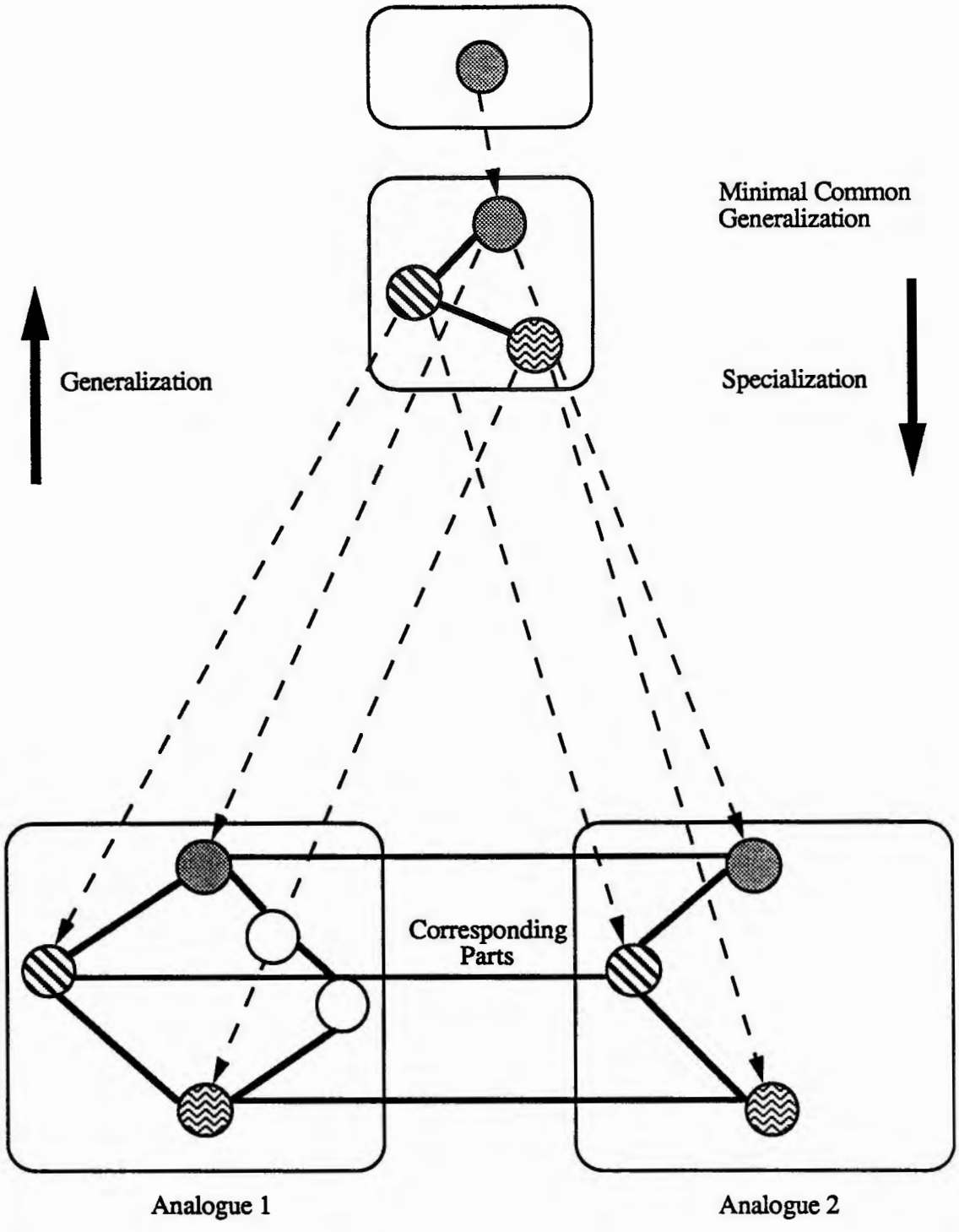


Figure 1 A General Framework for Analogy

can be restricted, either by addition of an individual to a generic concept or by replacing a concept with one of its subtypes from the concept lattice. Second, a conceptual graph can be specialized by making the graph more extensive. This amounts to adding more concepts and relations to a graph. Rules for when and how these specialization operations take place are specified by the conceptual graph model [Sowa, 1984]. These specialization rules and operations are not truth preserving but enforce constraints on sentence formation.

Generalization of conceptual graphs is truth preserving and if a graph u is a specialization of a graph v, written u <= v, then v is a generalization of u. Generalization of conceptual graphs defines a partial ordering called a generalization hierarchy. Graphs can be generalized in three ways:

1) Any subgraph is a generalization of the original graph;

2) Replacing a type label with a supertype generalizes a graph;

3) Deleting an individual from the referent field generalizes a graph.

These specialization and generalization operations have important uses in analogy formation and analogical reasoning.

Conceptual graphs were chosen as the knowledge representation scheme for the Analogical Tool because of their linguistic and logical basis. They are a semantically rich scheme which captures much of the fullness of natural language while maintaining necessary logical connections. The choice of conceptual graphs is based on several other reasons as well:

1) They have a mapping to logic and thus are compatible with any system based on logic.

2) Their straightforward mapping to and from natural language allows for a linguistic description of analogues.

3) They are computational in that they have well defined algorithms for manipulation of the graphs.

4) Their definition and associated operations contain elements which correspond naturally to the general framework for analogy.

The notions of "part-of", "generalization" and the "generalization ordering" relation of the general framework for analogy described above have corresponding elements in the conceptual graph model. Parts in the conceptual graph model consist of concepts, relations and the arcs between them. Generalization in the conceptual graph model happens in two ways, by climbing the type hierarchy and by taking subgraphs of an original graph. Generalizations are also partially ordered in the conceptual graph model. These corresponding elements define how the Analogical Tool forms and orders analogies which are seen as minimal common generalizations, and also allows for analogical inferencing to be performed within a single framework. Thus, instantiation of the general framework using conceptual graphs results in a computational tool which can be seen to both form analogies and perform analogical inferencing. The inferencing is not an add-on explicit part of the system, rather it is an integrated extension of the existing framework.

3 The Analogical Tool

3.1 Minimal Common Generalizations

In developing the general framework, it was noted that most applications of analogy prefer minimal common generalizations as the "stronger" analogies. The conceptual graph formalism contains a notion of minimal common generalization which directly corresponds. Figure 3 gives a simple example of three ordered common generalizations which can be formed from the two analogues shown.

"Sam thinks that the house has a kitchen and that Ivan believes that there is a cat in the kitchen"

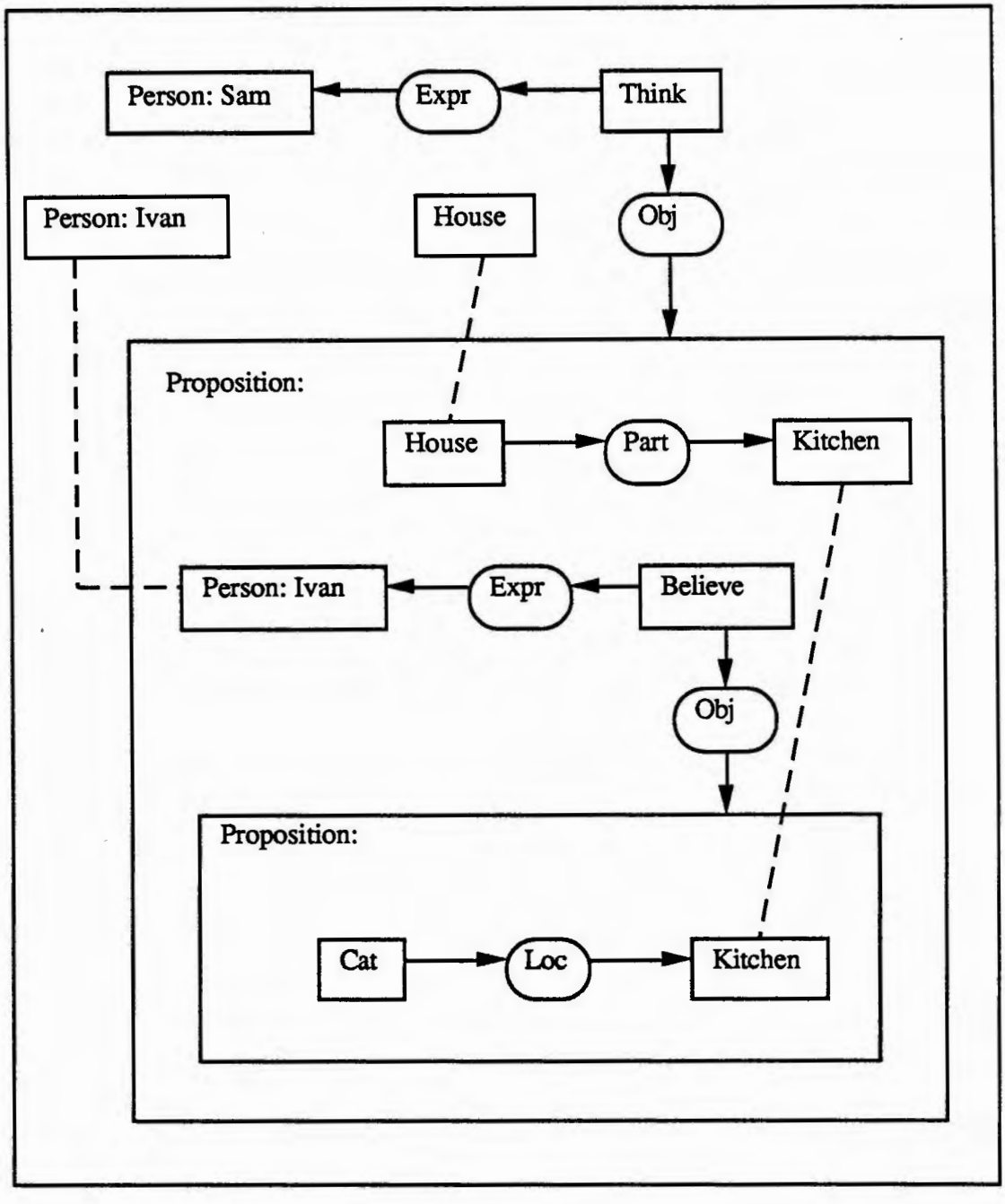


Figure 2 A Complex Sentence Represented in Conceptual Graphs

The lowest generalization on the page is the "stronger" minimal common generalization. It is the most specific of the three, first in concept restriction (from Animate to Person and from Act to Eat) and second by extent of the subgraph represented (it contains the maximum number of nodes and arcs). This minimal common generalization is the

generalization as low as possible in the generalization hierarchy - such that no other is more specific. These minimal common generalizations which the Analogical Tool derives:

1) Are maximally extensive since the generalization chosen for each of the corresponding concepts is the lower bound of all possible common generalizations.

2) Preserve linkages in the graphs which can be seen as minimal structural criteria for plausibility.

3) Are truth preserving, a necessary logical criterion for plausibility.

4) Minimize conflicts in the type hierarchy, a minimal semantic criterion for plausibility.

In forming the minimal common generalizations, the Analogical Tool uses four formation evaluations to search for the generalizations, and two ordering evaluations to establish which of these are minimal. Within the conceptual graph model, graphs are generalized in two ways. First, they are generalized by choosing subgraphs of an original graph and second, by using more general concepts according to the type hierarchy. The formation evaluations and ordering evaluations presented below correspond directly to these two types of generalization.

3.2 Formation Evaluations

There are four formation evaluations which work together to derive the common generalizations. Three of the four can be seen as working to decide what subgraphs are chosen for the resulting common generalization. These subgraphs must be common to both analogues, and they decide what parts of the analogues are put into correspondence. Any combination of subgraphs common to both analogues will form a generalization, but specific aspects of the conceptual graph model define what are the more plausible subgraphs. These aspects, which are the formation evaluations, also help to cut down on the computational complexity of the search by specifying that only certain subgraphs are considered for the resulting common generalization.

Same Relation Evaluation The first evaluation requires same relations between two pairs of potentially corresponding concepts. As shown in Figure 3 if [Girl] and [Person:Sue] are put into correspondence, then it is possible to put [Eat] and [Eat] into correspondence because they share the same (Agent) relation between them. If on the other hand Analogue 1 had contained [Girl] <- (Expr) <-[Think] instead, even though it would normally be possible to put [Girl] and [Person:Sue] in correspondence and also [Think] and [Eat] in correspondence (because they are both acts), using this evaluation, they cannot be put into correspondence because they no longer share the same relation between them. This evaluation is a result of the projection operator which requires that the same relation exist between connected corresponding concepts.

This evaluation defines one constraint on the subgraphs chosen for the common generalization. It is a very strong constraint which permits only agents to match to agents, objects to match to objects, etc.

Co-reference Links Evaluation Co-reference links define an equality relation between concepts. Thus, they define another structure which must be maintained when choosing subgraphs for the common generalization. If two concepts which both have co-referents are put into correspondence, then their co-referents can only be put into correspondence with each other - not with any other concepts. For example, if A=B and C=D where A, B, C and D are concepts in a conceptual graph and = represents the

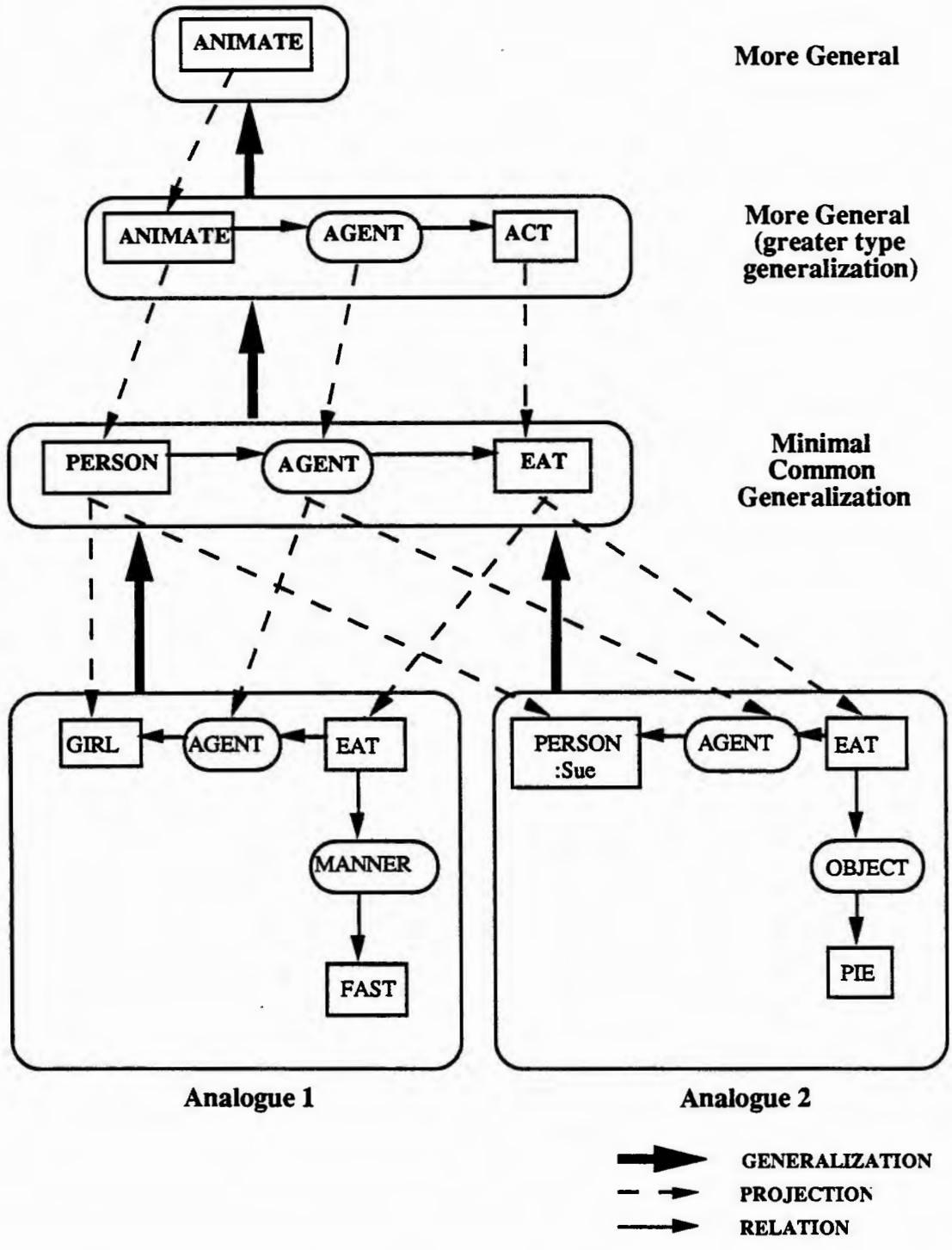


Figure 3 Forming Common Generalizations

co-reference link, then putting A and C into correspondence requires that B can only be put in correspondence with D or with nothing. Breaking this evaluation means the equality relation between concepts is being invalidated. The co-referent subgraph is a very strict evaluation that must be maintained.

Proposition Evaluation Proposition concepts which contain conceptual graphs in their referent fields define a *contains* relation between the proposition concept and the graphs within it. This relation defines yet another subgraph evaluation. This subgraph evaluation requires that if proposition concepts from two analogues are put into correspondence, then their respective contained graphs are candidates to be put into correspondence only with each other. The subgraph chosen here for the common generalization is the proposition concept with some or none of its contained graphs.

Comparable Concepts Evaluation This fourth evaluation conforms to the second way a graph can be generalized; that is by climbing the type hierarchy. Concepts are comparable if there exists a common generalization of them in the type hierarchy that is not the universal type. This evaluation uses the notion of type abstraction to determine if concepts are comparable and can be generalized. This evaluation works in conjunction with the part-of, subgraph evaluations by determining if specific concepts within a chosen subgraph can be generalized. Generalization using the type hierarchy also specifies particular general concepts for the resulting common generalization, as shown in Figure 3. There we see Person as a generalization for [Girl] and [Person:Sue] for example. In this way a common generalization results in the kind of general problem solving abstraction used by researchers such as Gick and Holyoak [1983] and Greiner [1985].

3.3 Ordering Evaluations

The formation evaluations are a consequence of specifying an analogy as a common generalization. Specification of "stronger", more plausible analogies as minimal common generalizations has implications for ordering them. The two ordering evaluations order the generalizations and conform to the generalization criteria for conceptual graphs in much the same way the formation evaluations do. The first ordering evaluation conforms to the subgraph generalization criterion. For minimal common generalizations, the subgraphs chosen are the maximally extensive or most specific common generalizations. The second ordering evaluation conforms to generalization using the type hierarchy. Minimal common generalizations are also those which choose the lowest common supertypes for each pair of corresponding concepts as defined by the type hierarchy.

Deriving minimal common generalizations according to the first ordering evaluation amounts to ordering the common generalizations by their *extent*. The second ordering evaluation, *semantic distance*, uses the distance climbed in the type hierarchy when searching for a common supertype for corresponding pairs of concepts.

Using these ordering evaluations, as the common generalizations are formed, it can be determined if they are minimal common generalizations so far. With a complete search of all analogies, the evaluations derive the set of minimal common generalizations or "stronger" analogies.

3.4 Analogical Inferencing

The Analogical Tool described thus far forms analogies which are characterized as minimal common generalizations. These analogies are useful for many purposes such as schema abstraction [Gick and Holyoak,1983]. Other uses of analogy require that further reasoning take place from the analogies. This reasoning is referred to as analogical inferencing.

Performing analogical inferencing consists of extending a target analogue by hypothesizing that new concepts, relations and arcs exist in it, based on the existence of similar concepts, relations and arcs in the source analogue. These inductive hypotheses result in the formation of a new minimal common generalization which contains a generalization of these new hypothesized parts and the source parts they are inferred from.

The formation evaluations used by the Analogical Tool can help determine which hypotheses to make and also determine which hypotheses are valid. Although parts could be hypothesized from either of the analogues to the other, the tool only hypothesizes parts as existing in the target analogue based on their existence in the source analogue.

The Extensibility Principle

When a choice must be made between two or more minimal common generalizations as to what inferencing is more plausible, the *extensibility principle* gives a resolution. If for instance one of the generalizations is not extensible at all and the other is, the extensible generalization is used because analogical inferencing can occur. If on the other hand, both are extensible, then the one which is the most extensible is preferred. Each of the extensions must still conform to the formation evaluations, but the most extensible generalization is the new minimal common generalization. We will still be deriving minimal common generalizations.

4 An Example from Gick and Holyoak - Schema Abstraction

The following example from Gick and Holyoak [1983] is discussed in detail to show how analogy formation takes place in the tool and how analogical inferencing proceeds. A result of analogy formation is production of a problem solving schema. Inferencing using this general problem solving schema can later be used to solve similar problems.

In Figure 4, Gick and Holyoak use as an example a military problem and an analogous radiation problem. In the military problem, a commander wants to capture a fortress using a large army but cannot send the entire army down one road because they will be noticed. The solution is to send small parts of the army along many roads simultaneously. These sentences are represented in the four conceptual graphs at the top of Figure 4. Co-referent links are shown by the very angled arcs connecting concepts with the same name. Context boxes representing propositions are the boxes which contain subgraphs within them.

In the analogous radiation problem, a doctor wants to use rays to destroy a tumor. The rays cannot all come from one direction without killing the patient. The analogous solution is to send low intensity rays from multiple directions simultaneously. These sentences are represented in the bottom four conceptual graphs containing dashed nodes in Figure 4.

These two situations as represented are given to the analogical tool, which forms the same analogy as that presented by Gick and Holyoak. The following concept correspondences are formed by the tool:

(Doctor - Commander) (Rays - Army) (Destroy - Capture) (Tumor - Fortress)

(Want - Want) (Possesses - Possesses) (Powerful - Large) (Go - Go)

(Direction - Road) (One - One) (WeakRays - Group) (Many - Many)

(Simultaneously - Simultaneously)

The tool also automatically forms an abstraction similar to that described by Gick and Holyoak, by using the concept hierarchy to find common generalizations of the concepts. The general problem solving principle formed by the tool is:

A Person wants to use a Mobile Entity to perform a Violent act on a Stationary Entity.

A Person has a Mobile Entity with an Attribute of Strength.

The Mobile Entity cannot go along one Path.

The Solution is for Parts of the Mobile Entity to Go along Many Paths Simultaneously

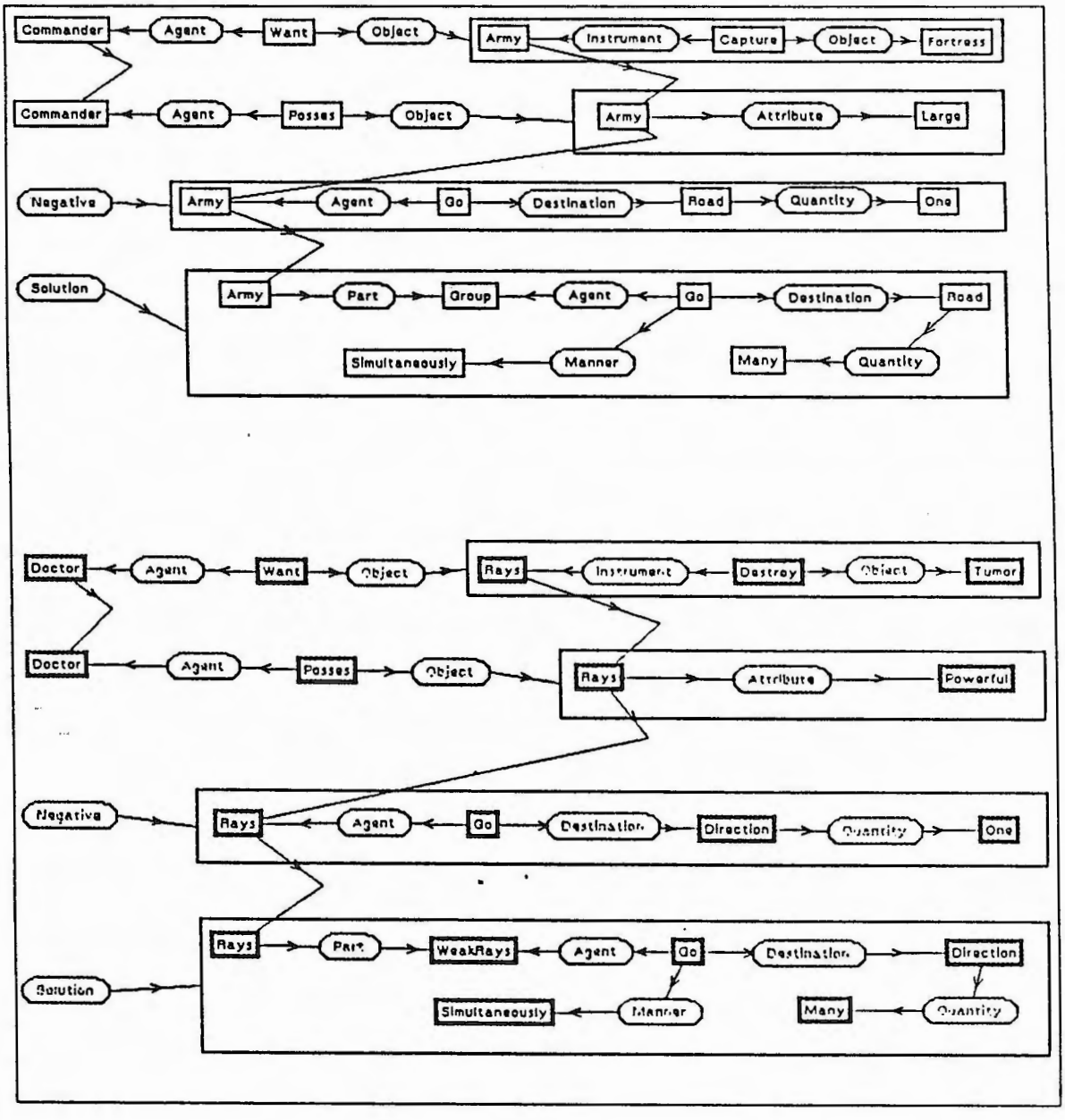


Figure 4 Gick and Holyoak - Schema Abstraction

This example can also be used to show how the tool can perform simple analogical reasoning. In this case the target analogue is the same with the exception that the solution to the radiation problem is not known. This is represented as an empty *Proposition* node attached to the *Solution* relation. When the tool reports the correspondences resulting from the best analogy formed, the *Solution* propositions from the target analogue is matched with the solution node of the abstracted schema. With substitution of corresponding terms inside the solution to the abstracted schema, a solution to the radiation problem is realized: to have parts of the rays go along many paths simultaneously.

5 Relevancy Limitations and Solutions

The Analogical Tool described thus far is a similarity based system and as such suffers from limitations due to relevancy issues. Drawing inferences from minimal analogies is an inductive process and thus potentially false. Use of the extensibility principle deals with some problems of relevancy and is similar to Gentner's [1983] use of the systematicity principle. But, analogical inferencing based on correspondence alone can be misleading if the major correspondences are not relevant to the objective of the inferencing. Russell [1987] defines one method of dealing with this problem using determinations. Determination rules are similar to dependency relations in database theories and are of the form: P determines Q. These rules are weaker than implications but serve as background knowledge which when added to the premises of an analogy, makes the conclusion follow soundly.

Current work on the Analogical Tool involves implementing the determinations rules using the type hierarchy. The type hierarchy appears to include much of the same information contained in the determination rules and a use of semantic distance over the type hierarchy can achieve the same effect. What we need to do when forming correspondences is to prefer and look for, correspondences which are semantically closer.

6 Summary

Analogy can be widely used in the knowledge acquisition cycle, in particular for schema abstraction and problem solving. This paper has described an analogical tool and shown its use in analogy formation, schema abstraction and analogical inferencing. Evaluation criteria have been derived for forming and ordering analogies by plausibility that relate naturally to operations on conceptual graphs.

References

[Gaines and Shaw, 1982] Brian Gaines and Mildred Shaw. Analysing Analogy. Trappl, R. Ricciardi, L. and Pask, G. Editors. Progress in Cybernetics and Systems Research Vol. IX. pp. 379-386. Washington: Hemisphere. 1982.

[Gentner, 1983] Dedre Gentner. Structure-Mapping: a Theoretical Framework for Analogy. Cognitive Science 7. 1983. pp. 155-170.

[Gentner and Seziorski, 1986] Dedre Gentner and M. Seziorski. Historical Shifts in the Use of Analogy in Science To appear in B. Gholson, W.R. Shadish, and A Graesser (Editors), Psychology of Science. 1986.

[Gick and Holyoak, 1983] Mary Gick and Keith Holyoak. Schema Induction and Analogical Transfer. Cognitive Psychology 15. 1983. pp.1-38.

[Hofstadter and Mitchell, 1988] Douglas R. Hofstadter and Melanie Mitchell. Concepts, Analogies and Creativity. Proceedings of CSCSI'88: Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence. pp.94-101. 1988.

[Indurkhya, 1985] Bipin Indurkhya. A Computational Theory of Metaphor Comprehension and Analogical Reasoning. PhD thesis, University of Massachusetts, 1985.

[Lakoff and Johnson, 1980] George Lakoff and Mark Johnson. The Metaphorical Structure of the Human Conceptual System. Cognitive Science 4. 1980. pp. 195-208.

[Leishman, 1989] Debbie Leishman. An Analogical Tool: Based on Evaluations of Partial Correspondences over Conceptual Graphs. Masters Thesis. University of Calgary. 1989.

[Lenat et al., 1986] Doug Lenat, Maryank Prakash and Mary Shepherd. CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. The AI Magazine. pp. 65-85.

[Murphy et al, 1963] Glenn Murphy, David J. Shippy and H. L. Luo. Engineering Analogies. Iowa State University Press. Ames, Iowa. 1963.

[Polya, 1954]. George Polya. Induction and Analogy in Mathematics. Volume I of Mathematics and Plausible Reasoning. Princeton University Press. Princeton. 1954.]

[Russell, 1987] Stuart Jonathan Russell. Analogical and Inductive Reasoning. PhD thesis. Stanford University. 1987.

[Sowa, 1984] John Sowa. Conceptual Structures. Addison Wesley Publishing Co. Reading Ma. 1984.

[Winston, 1981] Patrick Winston. Learning and Reasoning by Analogy. Communications of the ACM Vol. 23 No. 12 1980.

[Witten, & MacDonald, 1987] Ian Witten & Bruce MacDonald. Using Concept Learning for Knowledge Acquisition. Proceedings of the Second Knowledge Acquisition for Knowledge Based Systems Workshop. Banff, Alberta, Canada. 1987.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

EMPIRICAL KNOWLEDGE REFINEMENT IN NOISY DOMAINS

Ting-peng Liang
Department of Accountancy
University of Illinois at Urbana-Champaign
1206 South Sixth Street
Champaign, Illinois 61820

ABSTRACT

Knowledge acquisition is a bottleneck for developing expert systems. An initial knowledge base acquired from human experts often contains inconsistent rules. Knowledge refinement is a process by which a knowledge base can be modified to improve its performance. In this paper, we focus on the refinement of probabilistic rules. We first describe both theoretical and empirical anomalies in knowledge bases of probabilistic rules. Then, we present an empirical approach to refining these knowledge bases. We apply statistical tests to determine the significance of misclassifications and modify rules only if systematic errors are found. This can reduce the possibility of overfitting the case knowledge.

1. INTRODUCTION

The knowledge acquisition process, essential to the development of expert systems, consists of two phases: initial acquisition of the knowledge base, followed by its refinement. Due to human cognitive limitations in articulating knowledge, the initial knowledge base acquired from human experts frequently consists of errors. In addition, knowledge is dynamic; it evolves over time and needs continuous modification. Knowledge refinement is a process by which an initial knowledge base is verified, and plausible refinements to the rules in the knowledge base are identified and incorporated in order to improve its empirical accuracy.

A knowledge refinement process usually includes two steps: (1) anomalies of the knowledge base must be identified; and (2) proper actions must be taken to remove the anomalies. In previous research, several potential anomalies in knowledge bases such as redundancy, inconsistency, subsumption, and incompleteness have already been identified, and a few approaches to rule refinement have been proposed (see [Nazareth, 1989] for a review). Generally speaking, existing knowledge refinement methods fall into two categories: theoretical and empirical. Theoretical analysis focuses on checking the logical consistency and completeness of a knowledge base. The refinement program establishes a dependency table to represent the causal relationships in the knowledge base, locates logical anomalies found in the table, and generates advice for refinement accordingly (e.g., Ginzberg, 1986, 1988; Nguyen, et al, 1987; Suwa, Scott, and Shortliffe, 1982). Empirical analysis, on the other hand, applies case knowledge that includes cases with known conclusions to verify the validity of a knowledge base. Cases misclassified by the existing knowledge base trigger the rule refinement process that traces the reasoning process, identifies plausible refinements, and then generalizes or specializes certain rules to cover the mistakes (e.g., Ginzberg, Weiss, and Politakis, 1985, 1988; Laird, 1986; Lee and Ray, 1986; Politakis and Weiss, 1984).

In both lines of previous work, the domain to be modeled is assumed to be noise-free. In other words, the conclusion of a rule is either true or false. Little emphasis has been put on the effect of noise in the knowledge refinement process. Given that many problem domains

in the real world are noisy, conclusions derived from rules are often probabilistic, and some degree of misclassification is almost inevitable. As a result, the anomalies in the knowledge base must be redefined, and the process of knowledge refinement must be developed to accommodate the existence of noise. For example, two rules with the same left-hand-side and different conclusions are considered conflicting in a pure domain, but may be legal in a noisy domain as long as their combined probability is less than one.

The focus of this paper is to investigate some issues involved in refining knowledge in noisy domains. A knowledge refinement method taking into consideration the uncertainty in noisy domains to avoid ***"overfitting"*** the existing case knowledge is presented. It incorporates a statistical technique (p-test) into the rule refinement process to determine the significance of a misclassification and makes necessary generalization or specialization accordingly. Development of the method is part of the CRIS project on integrating statistical and machine learning techniques to improve the performance of knowledge acquisition (Liang, 1989). The remainder of the paper is structured as follows. Section 2 defines knowledge base anomalies in noisy domains. Section 3 presents the new approach to refining a knowledge base. Section 4 provides conclusions.

2. KNOWLEDGE BASE ANOMALIES IN NOISY DOMAINS

Inconsistency and incompleteness have been identified by previous research as the major problems in a knowledge base. An inconsistent knowledge base indicates that some rules can be removed without reducing its overall performance. Generally speaking, rule inconsistency includes redundancy, conflict, subsumption, unnecessary IF conditions, and cycles in the knowledge base. An incomplete knowledge base indicates that some necessary rules are missing. It includes missing or illegal attribute values and unreachable conclusions. Discussions of these anomalies in a pure domain can be found in Nazareth, 1989; Nguyen, et al., 1987; and Suwa, et al., 1982.

In noisy domains, rules representing the knowledge usually include uncertainty measures to tolerate various noises. In other words, the conclusion of a rule is no longer either true or false. There is a probability that the conclusion of the rule is false, even though all of its left-hand-side conditions are true. These rules are called probabilistic rules, which are different from the deterministic rules used to represent perfect knowledge. The inclusion of uncertainty measures in rule representation has an effect on both the inference process of an expert system and the detection and correction of anomalies in a knowledge base. In particular, it has a significant impact on consistency checking. We shall redefine the anomalies that may exist in a probabilistic knowledge base below.

(1) Redundancy

In a pure domain, two rules are redundant if, given the same input, they always reach the same conclusion. Symbolically, two rules, "$C(X) \rightarrow R(X)$" and "$C(Y) \rightarrow R(Y)$," are said to be redundant, where C is the condition of a rule, R is the conclusion of a rule, and X and Y are variables to be instantiated by input values.

In a noisy domain, however, this definition needs to be modified to incorporate the uncertainty measure. That is, two rules are redundant if they have the same conditions, conclusions, and probabilities. In other words, two rules, "$C(X) \rightarrow R(X)$, $P = p_0$" and "$C(Y) \rightarrow R(Y)$, $P = p_1$," are redundant if $p_0 = p_1$, where p_0 and p_1 are the probabilities associated with these two rules. If two rules have the same condition and conclusion but different uncertainty values (i.e., $p_0 \neq p_1$), they are considered conflicting rules.

(2) Conflict

In a pure domain, two rules are conflicting if, given the same input, they reach conflicting conclusions. For example, two rules, "C(X) → R(X)" and "C(X) → ~R(X)," are conflicting because ~R(X) (i.e., not R(X)) is the contradiction of R(X).

In a noisy domain, there are two types of conflict. First, as described in rule redundancy, two rules may have the same set of inputs and conclusions but different probabilities. They are conflicting because only one of them can be true. In the second type of conflict, two rules derive different conclusions from the same set of inputs and their combined probability does not equal one. It should be noted that probabilistic rules allow multiple conclusions to be reached from the same set of inputs as long as their combined probability equals one. Symbolically, we say that two rules, "C(X) → R(X), P = p_0" and "C(X) → ~R(X), P = p_1," are conflicting if $p_0 + p_1 \neq 1$.

(3) Subsumption

In a pure domain, a rule is said to be subsumed by another if the two rules have the same conclusion, but one contains additional conditions to restrict its success. Symbolically, the rule, "C(X) ∧ D(X) → R(X)" is subsumed by the rule, "C(X) → R(X)," because whenever the former succeeds, the latter also succeeds.

In a noisy domain, different rule probabilities indicate different levels of specialization and hence can reduce the likelihood of rule subsumption. A rule is said to be subsumed by another if these two rules have the same conclusion and the same probability to be true, but the former contains extra conditions. That is, the rule "C(X) ∧ D(X) → R(X), P = p_0" is subsumed by the rule "C(X) → R(X), P = p_1," if $p_0 = p_1$.

(4) Unnecessary IF conditions

In a pure domain, a knowledge base is said to contain unnecessary IF conditions if two rules have the same conclusion, but some of their conditions are contradictory. Symbolically, two rules, "C(X) ∧ D(X) → R(X)" and "C(X) ∧ ~D(X) → R(X)," include unnecessary IF conditions because D(X) and ~D(X) are contradictory. We can easily combine these two rules into a single rule "C(X) → R(X)" without affecting the validity of the knowledge base.

As in rule subsumption, the unnecessary IF conditions exist in probabilistic rules only when the probabilities of these rules are the same. That is, a knowledge base is said to contain unnecessary IF conditions if two rules, "C(X) ∧ D(X) → R(X), P = p_0" and "C(X) ∧ ~D(X) → R(X), P = p_1," have the same probability (i.e., $p_0 = p_1$). Otherwise, D(X) cannot be eliminated by combining these two rules.

(5) Cycles

In a pure domain, a set of rules is said to be cyclic if every rule in the set has at least one conclusion which is also the input condition of another rule in the set. For example, rules "C(X) → Q(X)," "Q(X) → R(X)," and "R(X) → C(X)," are cyclic.

In a noisy domain, the existence of a cycle is determined by both the chaining relationship and the strength (probability) of the chain. The strength of a chain is the product of the probabilities of its component rules. If the strength is higher than a predetermined threshold probability considered significant by the inference engine, then the cycle exists. Otherwise, the inference engine can eliminate the cycle automatically. Symbolically, rules "C(X) → Q(X), P = p_0," "Q(X) → R(X), P = p_1," and "R(X) → C(X), P = p_2" are

cyclic, if $(p_0 \times p_1 \times p_2) > p_t$, where p_t is the threshold probability used by the inference engine to determine the significance of an evidence.

Given different definitions of knowledge anomalies in noisy domains, the refinement of a knowledge base must consider not only the conditions and conclusions of the rules, but also their uncertainties. All of the anomalies described above can be identified and removed by conducting a theoretical analysis that checks all logical relationships in the knowledge base. No empirical evidence is necessary. Therefore, they are called ***theoretical anomalies.*** In addition to these theoretical anomalies, there are situations in which a theoretically perfect knowledge base may be empirically inaccurate. The anomalies that can only be identified by empirical evidence are called ***empirical anomalies.***

Empirical anomalies occur when (1) the attributes have high degrees of correlation or (2) the knowledge base is not an accurate representation of the problem solving knowledge. In the first situation, attributes with a high degree of correlation usually have overlapping power of discrimination. This means that some of them may be removed without reducing the performance of the knowledge base. If the height and weight of a person are to be used to predict gender, for example, removing one of them may not reduce the prediction accuracy significantly because they are highly correlated. Empirical anomalies resulting from attribute correlation usually include two types: ***empirical redundancy and subsumption***[1].

Two rules, "$C(X) \rightarrow R(X)$, $P = p_0$" and "$D(X) \rightarrow R(X)$, $P = p_1$," are empirically redundant, if for all known cases they have the same true or false value and $p_0 = p_1$. Empirical redundancy occurs when two attributes are perfectly correlated (i.e., the correlation coefficient of their data equals one). The rule "$C(X) \rightarrow R(X)$, $P = p_0$" is said to be empirically subsumed by the rule "$D(X) \rightarrow R(X)$, $P = p_1$," if all known cases proven true by the former rule are also proven true by the latter and $p_0 < p_1$.

In the second situation, the noise in a domain may lead to the development of inaccurate rules. For example, if a rule with a probability of 70% reaches false conclusions in 30 of 50 cases, then it is very likely that the rule is ***empirically invalid*** and needs to be modified.

In order to correct theoretical anomalies in a knowledge base, existing methods of theoretical analysis need to be modified to incorporate steps that check rule uncertainty. In other words, the revised approach must include three steps. First, a dependency table including conditions, conclusions, and rule uncertainties must be established. Second, rule redundancy and consistency based on rule conditions and conclusions must be checked without considering uncertainty (i.e., using existing methods applicable to deterministic rules). Third, if anomalous rules are found, uncertainty information must be checked for further confirmation. Empirical anomalies must be corrected by using previous case knowledge to evaluate and modify the existing rules. In the following section, an approach to empirical rule refinement, called PR (Probabilistic Refiner), is presented.

---

[1]The negative effect of empirical redundancy is that the correlated attributes may be overweighted by the inference engine and hence lead to an inaccurate conclusion. This is particularly true when the inference engine uses a cumulative approach (such as the certainty factor approach) to assess the uncertainty of the final conclusion. It is, however, useful when some data may be unknown or unreliable. In this case, redundant rules can be a good substitute or can be used for double-checking. Therefore, a good alternative is to remove the redundant rules from the primary knowledge base and store them in a secondary one. They will be activated for double-checking if their correlated rule in the primary knowledge base fires.

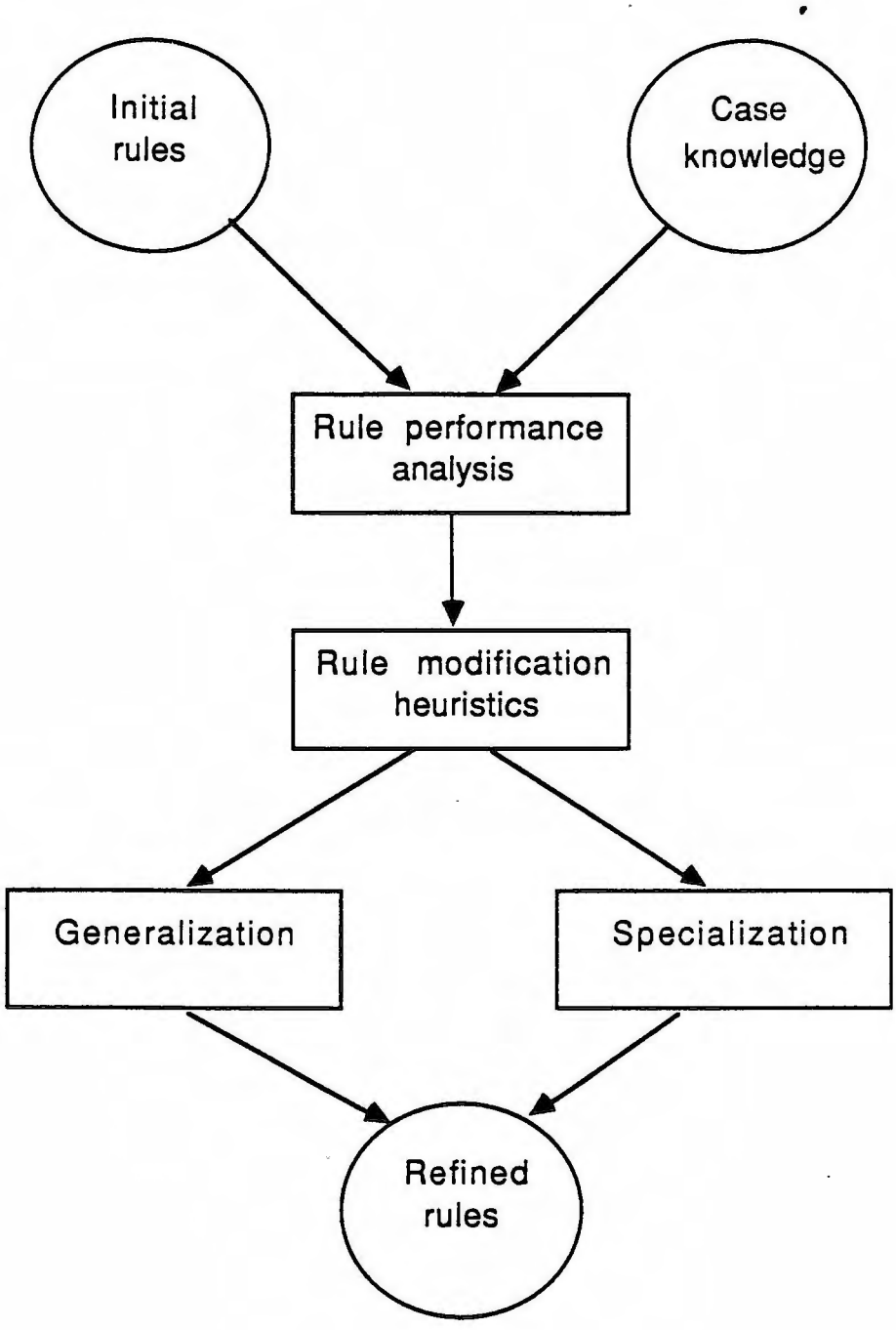


Figure 1. Empirical refinement of knowledge

3. AN APPROACH TO EMPIRICAL REFINEMENT OF A KNOWLEDGE BASE

Empirical refinement is a process by which the rule refinement mechanism analyzes cases with known conclusions to improve the accuracy of the knowledge base. It is important because a set of theoretically perfect rules may be empirically inaccurate. In general, theoretical analysis can be used to verify whether a knowledge base is organized correctly, whereas empirical analysis can be used to validate whether a knowledge base accurately represents the problem solving knowledge. A typical empirical mechanism must include modules for performance analysis and for rule modification, as shown in Figure 1. The performance analysis module evaluates the performance of each rule by examining its significance of misclassification and identifies empirically invalid rules accordingly. The rule modification module revises empirically invalid rules with generalization or specialization. Generalization is an operation that makes a rule easier to satisfy, whereas specialization is an operation that makes a rule more difficult to satisfy. Furthermore, empirically redundant or subsumed rules may also be removed from the primary knowledge base to the secondary one to improve inference efficiency. In this Section, methods for refining invalid rule and for removing redunancy are discussed in 3.1 and 3.2, respectively.

3.1 Refinement of Invalid Rules

Given that a certain level of misclassification is considered to be inevitable and tolerable in a noisy domain, the first step to knowledge refinement is to determine whether the misclassification level is significant enough to trigger a refinement (i.e., whether the rule is invalid). This requires the application of statistical tests. If the level is proven statistically significant, then operations for refining the conditions, conclusions, or probability of the rule are activated and implemented. Otherwise, no refinement will be performed.

Based on this understanding, the rule refinement process of PR includes several steps as shown in Figure 2. First, the initial knowledge base is applied to derive a conclusion for each test case. This conclusion is then compared with the known conclusion (which may be provided by human experts or available in archives). If they are the same, then the knowledge base correctly interprets the case. The case will be identified as a hit and stored in the case base. Otherwise, the system identifies the rules leading to the misclassification, retrieves cases previously misclassified by the same rule, and then tests the significance of the accumulated misclassification. If the error is proven to exceed the tolerance range of the rule (i.e., statistically significant), then the rule modification module will be activated. Otherwise, the case is identified as a miss and its relevant information is stored in the case base.

When the rule modification module is triggered, it performs two major functions: it generates plausible refinement and selects optimal refinement. The refined rules are then stored in the knowledge base to replace the original ones; the cases will be reclassified based on the refined rules and stored properly in the case base. In this paper, the first step, i.e., applying the initial knowledge base to a test case, is omitted because it is the same as applying expert systems to problem solving. Methods for testing significance, generating plausible refinement, and selecting optimal refinement are described in the following sections.

3.1.1 Test of Significance

The test of significance determines whether a misclassification is still within the tolerable range. Refinement is necessary only if the error is systematic (i.e., non-random). This test is important because it prevents the system from overfitting existing cases in a noisy

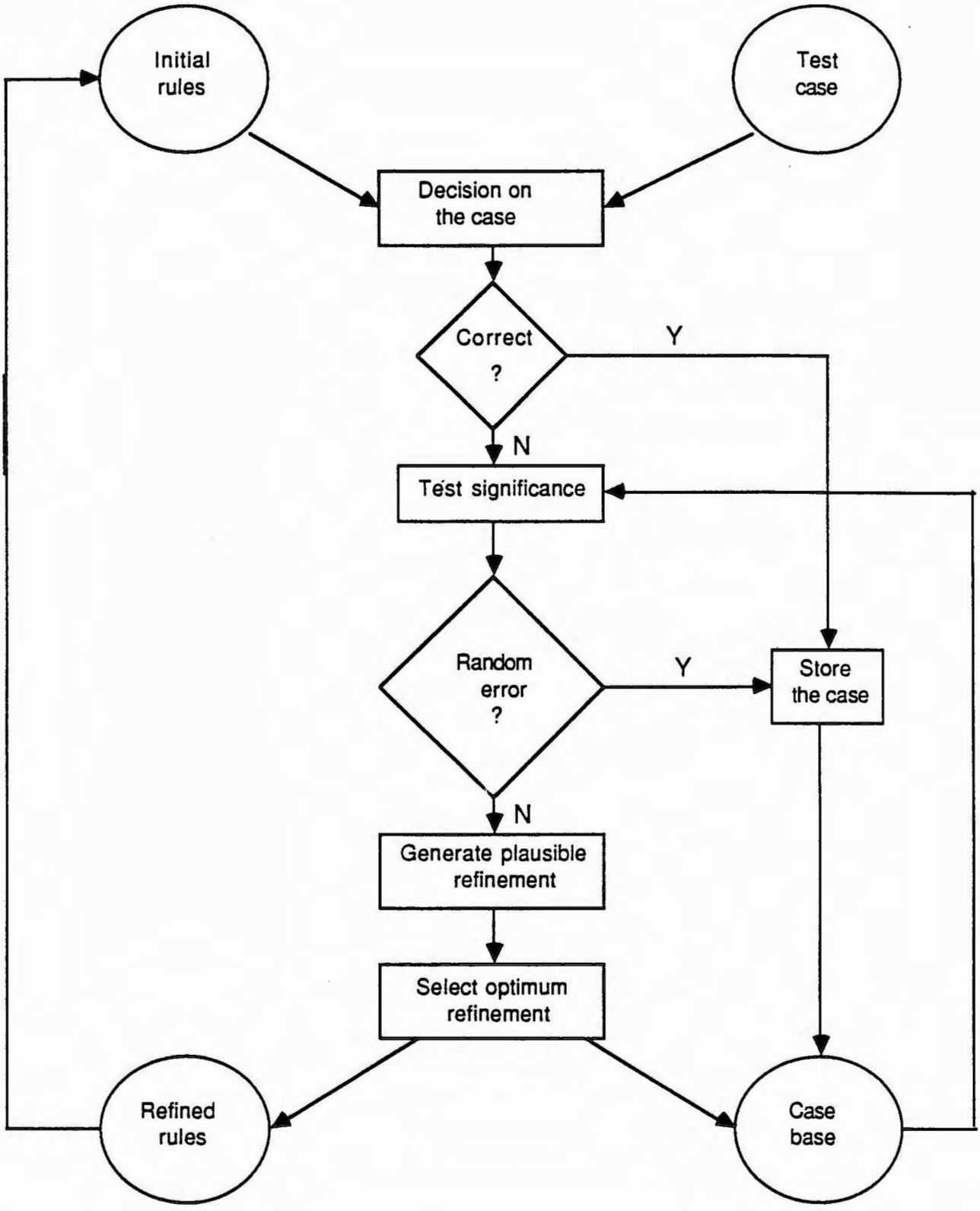


Figure 2. Process for refining probalistic rules

domain.

Since the probability associated with a rule represents the likelihood that the rule is correct, a p-test that determines whether a probability equals a certain value is considered appropriate for testing the significance of the error. For example, with the rule "$C(X) \rightarrow R(X)$, $P = p_0$," we assume the rule is correct at the probability p_0 and then try to prove that the actual probability of misclassification is significantly different from p_0. If the proof succeeds, then the error is considered systematic and the rule must be modified. Procedures for a p-test include the following:

(1) Calculation of the actual probability of misclassification, p_a.

$$p_a = \frac{\text{\# of cases correctly classified by the rule}}{\text{Total \# of cases classified by the rule}}$$

(2) Calculation of the critical value for testing significance, CV.

$$CV = \frac{| p_a - p_0 |}{\sqrt{p_0(1-p_0)/N}}$$

Where: N = total number of cases classified by the rule.

(3) If $CV \geq |Z(p_0)|$, then we conclude that the original hypothesis that the misclassification is a random error is rejected and the rule must be refined. Otherwise, the hypothesis is accepted and no refinement is needed. $Z(p_0)$ is the value of standard normal distribution whose cumulative probability is p_0. If N is small, then the Z-distribution can be replaced by a t-distribution and $Z(p_0)$ is replaced by $t(N-1, p_0)$.

For example, consider the following financial loan application rule:

IF annual income is greater than 25,000, and
the applicant owns a house
THEN approve the loan
WITH P = 0.95.

If the rule has misclassified four in fifty cases, then, following the previous procedures, P_a is 0.92 and CV can be calculated as follows:

$$CV = \frac{| 0.92 - 0.95 |}{\sqrt{0.95(0.05)/50}} = 0.973$$

The value of Z(0.95) is 1.645. Since $CV < Z(0.95)$, the misclassifications are considered to be random errors and no modification is necessary. However, if the number of misclassified cases increases to six out of fifty, then, by the same procedures, we can calculate $P_a = 0.88$, $CV = 2.27$. Because $CV > Z(0.95)$, the rule must be refined.

In addition to the statistical test of significance, another important issue at this stage is to determine which rule is responsible for a misclassification when multiple rules are involved in the inference process. This is called the blame (or credit) assignment problem (Grefenstette, 1988; Sleeman, Langley, and Mitchell, 1982). Although it is sometimes possible to locate the exact rule responsible for a misclassified case, it may not be necessary to do so in our case, because the misclassification may be due to data error. Therefore, PR adopts an

approach that focuses on finding the rule proven to be invalid, rather than on finding a single rule responsible for a particular case. In other words, when multiple rules are involved in a decision, each rule is evaluated separately by considering all cases it correctly classifies and misclassifies. Those rules proven significantly wrong are revised. The following example illustrates the method.

Suppose there are four rules in a knowledge base, whose performance is shown in Table 1 (the hit column indicates the number of cases classified correctly, whereas the miss column indicates the number of cases misclassified by the rule). When an additional case is found misclassified by rules R_1 and R_3, the total number of misclassified cases increases from seven to eight. Since R_1 and R_3 are evaluated separately, PR first retrieves all information related to R_1. The total number of hits is 68 and the total number of misses is 14, which results in an error ratio of 14/80. Similarly, the error ratio of R_3 is 12/90. Statistical tests are then performed on these two rules separately to determine whether a refinement is necessary. If neither result is significant, no refinement is necessary. If one of them is significant, then the rule must be refined. If both are significant, there are two possible refinement strategies. The conservative strategy is to refine the rule with the higher error ratio and ignore the other. The liberal strategy is to refine both. Therefore, when multiple rules are involved in a decision, the result of performance analysis may vary from revising no rule to revising all rules.

Rule ID	Hit	Miss	Total
R_3	18	2	20
R_4	17	3	20
$R_1 + R_3$	42	7	49
$R_1 + R_4$	24	6	30
$R_2 + R_3$	18	2	20
$R_2 + R_4$	27	3	30

Table 1. Sample Performance Data

3.1.2 Generation of Plausible Refinement

There are two basic operations for refinement of a rule that is determined to be wrong: generalization and specialization. They can be performed on the left-hand side condition, the probability of the rule, or both. For example, the previous financial loan application rule can be generalized by (1) increasing the probability from 0.95 to 0.98, (2) reducing the annual income value from $25,000 to $20,000, or (3) eliminating the requirement that the applicant must own a house. Specialization works in the opposite way. For example, we may specialize the sample rule by (1) decreasing the probability from 0.95 to 0.88, (2) raising the annual income requirement from $25,000 to $30,000, or (3) adding extra requirements, such as previous default records, to the left-hand side of the rule.

For each rule to be refined, PR first generates plausible alternatives and then selects the best one for implementation. The alternative generation process includes the following steps:

(1) When p_a (the actual performance) is lower than p_0, which indicates that the accuracy of the rule is overestimated, a specialization process is activated. Otherwise, a generalization is considered.

(2) In a specialization (generalization) process, the first (and probably the most straightforward) alternative is to replace the original probability p_0 with the actual ratio p_a.

(3) Then, the system establishes more alternatives by examining how the hurdle value associated with each attribute can be increased (decreased) to cover the misclassified cases.

(4) Finally, the system examines the attributes not included (included) in the rule to investigate the effect of adding (removing) them into (from) the rule. If there is more than one attribute to be considered, at least one alternative can be generated for each attribute. Sometimes, more than one attribute may be considered simultaneously.

For example, suppose that six of fifty cases are misclassified by the loan application rule and a refinement is found desirable. We further assume that there are two attributes not included in the rule which are usually considered to have some effect on a loan decision: previous default record and collateral. Since p_a (0.88) is lower than P_0 (0.95), a specialization is considered appropriate. Application of the above procedures may generate the following five plausible alternatives:

(a) IF annual income is greater than 25,000, and
the applicant owns a house
THEN approve the loan
WITH P = 0.88. (* Change probability *)

(b) IF annual income is greater than 30,000, and
the applicant owns a house
THEN approve the loan
WITH P = p_1. (*Change income value *)

(c) IF annual income is greater than 25,000, and
the applicant owns a house, and
the applicant has no previous default
THEN approve the loan
WITH P = p_2. (* Add an attribute *)

(d) IF annual income is greater than 25,000, and
the applicant owns a house, and
collateral is OK
THEN approve the loan
WITH P = p_3. (* Add an attribute *)

(e) IF annual income is greater than 25,000, and
the applicant owns a house, and
the applicant has no previous default, and
collateral is OK
THEN approve the loan
WITH P = p_4. (* Add two attributes *)

The probabilities, p_1, p_2, p_3, and p_4, are calculated by dividing the number of cases correctly classified by the proposed rules by the total number of cases (it is 50 in this

example).

3.1.3 Selection of the Optimum Refinement

For each generated alternative, an evaluation process is applied to determine (1) how many cases are correctly classified by the proposed rule, and (2) whether the improvement is statistically significant. The reason that a statistical test is required at this stage is that we would like to minimize the unnecessary rule modification due to the noisy nature of the domain.

The system first applies each of the proposed rules to all cases to determine its empirical performance. Please note that the alternative rule (a) in the previous section serves as a default for the refinement process. In other words, if other alternatives are proven unsatisfactory, then the default rule will be chosen. Since the only change it contains is a new probability, the modification does not reduce the number of misclassified cases, but it reflects the uncertainty of a conclusion more accurately. Table 2 lists the sample performance of the alternatives proposed in Section 3.1.2.

Rule	Hit	Miss	Total
(a)	44	6	50
(b)	45	5	50
(c)	48	2	50
(d)	47	3	50
(e)	48	2	50

TABLE 2. Performance of Alternatives

Based on the data in Table 2, we can calculate the probabilities associated with candidate rules (b)-(e) and determine whether they are significantly different from the default (i.e., alternative (a) with P = 0.88). We still use the p-test described previously.

(1) For alternative (b), $p_1 = 45/50 = 0.90$, and

$$CV = \frac{|\ 0.90 - 0.88\ |}{\sqrt{0.88(0.12)/50}} = 0.44.$$

(2) For alternative (c), $p_2 = 0.96$, and

$$CV = \frac{|\ 0.96 - 0.88\ |}{\sqrt{0.88(0.12)/50}} = 1.74.$$

(3) For alternative (d), $p_3 = 0.94$, and

$$CV = \frac{|\ 0.94 - 0.88\ |}{\sqrt{0.88(0.12)/50}} = 1.30.$$

(4) For alternative (e), $p_4 = 0.96$, and

$$CV = \frac{| 0.96 - 0.88 |}{\sqrt{0.88(0.12)/50}} = 1.74.$$

Since Z(0.88) = 1.175, the improvement of alternative (b) is considered statistically insignificant. It is therefore eliminated from the list for further consideration. In choosing among the plausible alternatives proven statistically significant, the system adopts the following criteria to select the optimum refinement.

1. If more than one alternative is available, select the one with the highest probability value.

2. If more than one alternative ties at the highest probability, choose the one with fewer left-hand conditions.

According to these two guidelines, alternative (c) in the previous example is considered the optimum refinement and the refined rule is

IF annual income is greater than 25,000, and
the applicant owns a house, and
the applicant has no previous default
THEN approve the loan
WITH P = 0.96.

3.2 Elimination of Redundant Rules

In addition to refining invalid rules, empirically redundant or subsumed rules are also removed from the primary knowledge base because they may lead to the undesired overweighting of certain attributes. Since the empirically redundant or subsumed rules are not logically dominated by the existing knowledge base, they may be found useful later. Therefore, the removed rules are stored in a secondary knowledge base[2]. This process includes (1) constructing a rule performance table, (2) checking the cases covered by each rule to identify redundancy and subsumption, and (3) removing redundant or subsumed rules to simpify the knowledge base.

A rule performance table includes four types of information: rule identification, rule uncertainty, cases correctly interpreted by the rule (hit), and cases misinterpreted by the rule (miss). Based on the table, empirical redundancy and subsumption can be identified. If two rules have exactly the same sets of hits and misses, then they are redundant. If the hit and miss sets of one rule are subsets of another rule, then the first rule is subsumed by the latter one.

Once redundant and subsumed rules are identified, they are removed from the primary knowledge base. For empirically redundant rules, the one with the highest probability remains in the knowledge base and the others are removed. All rules empirically subsumed by other rules are removed. This empirical refinement process can improve the efficiency of the knowledge base (e.g., reduce the size of the knowledge base) without decreasing its performance.

In developing a knowledge base for predicting firm bankruptcy, for example, twenty rules

[2]See footnote 1 for the pros and cons of having empirically redundant rules removed.

were initially included in the knowledge base. After a test on twenty cases, a rule performance table, as shown in Table 3, was established.

ID[1]	P[2]	Hits[3]	Misses[3]
R1	0.62	1,2,3,4,5,6,7,8,9,10	12,14,16,17, 19,20
R2	1.0	11,13,15,18	–
R3	0.79	1,2,3,4,5,6,7,8,9	12,18
R4	0.79	11,13,14,15,16,17, 19,20	10
R5	0.82	1,2,5,6,7,8,9	18
R6	0.85	1,2,5,6,7,8,9	–
R7	0.82	11,13,14,19,20	10
R8	0.86	11,13,14,19,20	10
R9	0.68	11,14,15,16,18,19	3,5,7,10
R10	0.68	1,2,4,6,8,9	12,13,17,20
R11	0.78	11,15,16,18,19	10
R12	0.83	11,15,16,18,19	10
R13	0.68	13	–
R14	0.83	13	–
R15	0.65	1,5,6,7,8,9,10	11,14,15,20
R16	0.65	12,13,16,17,18,19	2,3,4
R17	0.67	5,6,7,8,9	11
R18	0.67	5,6,8	11
R19	0.79	12,13,16,19	3,4
R20	0.86	13,16	4

Note: 1. ID = rule identification in Appendix 2.

2. P = probability of the rule.

3. The numbers in the hits and misses columns are the IDs of the training cases.

Table 3. A Rule Performance Table

The empirical anomalies shown in the table include the following.

(1) R7 and R8 are redundant and both are subsumed by R4. Therefore, R7 and R8 can be removed to the secondary knowledge base.

(2) R11 and R12 are redundant. R11 can be removed because it has lower probability.

(3) R13 and R14 are redundant. R13 can be removed.

(4) R18 is subsumed by R17 and can be removed.

(5) R20 is subsumed by R19 and can be removed.

After the refinement, six rules were removed from the knowledge base, which reduced its size by 30%.

4. CONCLUSIONS

In order to model the decision knowledge in a noisy domain, probabilistic rules that tolerate random errors are often used. The inclusion of uncertainty information in the knowledge base requires redefinition of anomalies and development of different knowledge refinement methods.

In this paper, we first defined theoretical and empirical anomalies in a probabilistic knowledge base. Then, we presented an empirical approach to refining probabilistic rules, which extended existing methods on empirical refinement. The new approach includes two major modules. The first module detects empirically invalid rules and suggests the optimum modification. It consists of procedures for testing the significance of misclassification, generating plausible alternatives, and selecting the optimum refinement. The second module detects and removes empirically redundant and subsumed rules from the primary knowledge base. Use of this empirical approach to rule refinement, along with a theoretical analysis that checks logical consistency and completeness, provides a useful way to improve the quality of a knowledge base.

5. REFERENCES

Ginzberg, A. (1988), ***Automatic Refinement of Expert System Knowledge Base***, Pitman, London, UK.

Ginzberg, A. (1988), "Knowledge Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency and Redundancy," ***AAAI-88 Proceedings***, St. Paul, MN, pp. 585-589.

Ginzberg, A., Weiss, S. M., and Politakis, P. (1988), "Automatic Knowledge Base Refinement for Classification Systems," ***Artificial Intelligence***, 35, pp. 197-226.

Grefenstette, J.J. (1988), "Credit Assignment in Genertic Learning Systems," ***AAAI-88 Procededings***, St. Paul, MN, pp. 596-600.

Lee, W. and Ray, S. (1986), "Rule Refinement Using the Probabilistic Rule Generator," in ***AAAI-86 Proceedings***, Philadelphia, PA, pp. 442-447.

Laird, P. D. (1986), "Inductive Inference by Refinement," ***AAAI-86 Proceedings***, Philadelphia, PA, pp. 472-476.

Liang, T. P. (1989), A Composite Approach to Inducing Knowledge for Expert Systems Design, BEBR Faculty Working Paper No. 89-1534, College of Commerce and Business Administration, University of Illinois, Champaign.

Nguyen, T. A., Perkins, W. A., Laffey, T. J., and Pecora, D. (1985), "Checking an Expert Systems Knowledge Base for Consistency and Completeness," in ***Proceedings of the Ninth International Joint Conference on Artificial Intelligence***, pp. 376-378.

Nguyen, T. A., Perkins, W. A., Laffey, T. J., and Pecora, D. (1987), "Knowledge Base Verification," ***AI Magzine***, Summer, 8:2, pp. 69-75.

Politakis, P. and Weiss, S. (1984), "Using Empirical Analysis to Refine Expert System Knowledge Bases," ***Artificial Intelligence***, Vol. 22, pp. 23-48.

Nazareth, D. L. (1989), "Issues in the Verification of Knowledge in Rule-based Systems," ***Int. J. Man-Machine Studies***, 30, pp. 255-271.

Sleeman, D., Langley, P. and Mitchell, T.M. (1982), "Learning From Solution Paths: An Approach to the Credit Assignment Problem," ***AI Magzine***, Spring, pp. 48-52.

Suwa, M., Scott, A. and Shortliffe, E. (1982), "An Approach to Verifying Completeness and Consistency in a Rule-based Expert System," ***AI Magzine***, 3:3, Fall, pp. 16-21.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A Methodology for Knowledge Acquisition in a Group Decision Support System Environment

Yihwa Irene Liou*
E. Sue Weber**
Jay F. Nunamaker, Jr.**

*Information and Quantitative Sciences Department
Robert G. Merrick School of Business
University of Baltimore
Baltimore, MD 21201

**Department of Management Information Systems
College of Business and Public Administration
University of Arizona
Tucson, AZ 85721

ABSTRACT

Typically, a Group Decision Support System (GDSS) environment provides computer and group-process support to individuals working together toward common goals. Such an environment characterized in this paper was used to facilitate the acquisition of knowledge from a group of experts. A methodology for elicitation of knowledge from multiple experts is presented and described in detail. Results of a field study which utilized the methodology to develop an expert system for an information center (IC) are discussed.

1 INTRODUCTION

Knowledge acquisition can be defined as a process of extracting and organizing the knowledge of several sources, usually human experts, so that it can be used in a computer program (Waterman, 1986). Because an expert system derives its power from the knowledge it possesses rather than from the particular formalisms and inference schemes it uses (Feigenbaum, 1977), this extraction and transfer of problem-solving expertise from knowledge sources to computer programs is the most critical task that knowledge engineers undertake. Consequently, one of the first important tasks that a knowledge engineer must accomplish is to identify appropriate domain experts who should be involved in the knowledge acquisition process.

In most organization, few substantial problems can be managed by single individuals. Thus, as extensive and organized as a single expert's knowledge might be, any expert system that derives from one individual's lines of reasoning or understanding will most likely be limited in some respect. In contrast, those expert

systems that embody knowledge of a group of experts are likely to function far more effectively in most situations (LeClair, 1985). Additionally, when no single individual completely understands a given domain and diverse subsets of knowledge are required, knowledge acquisition from multiple experts is a necessity. However, working with a group of experts is far from easy, particularly when individuals' domains of expertise overlap. For instance, besides differences in each individual's problem-solving capabilities and their abilities to explain "how" problems are solved, there can be conflicts between their problem-solving methods, interpersonal communication difficulties due to different professional orientations, and complications in integrating multiple views into one representation scheme.

Because the involvement of multiple experts in the development of expert systems has seldom been discussed in the literature, many of these issues have not been thoroughly explored. Some researchers have viewed the use of multiple experts in the knowledge acquisition process as a means to better understand the domain (Mittal and Dym, 1985). Others recommend that one participant be identified as a primary expert (Prerau, 1985) and group discussions be employed to interpret alternative points of view and problem solving methods (Chorafas, 1987). A method using the Expertise Transfer System (ETS) to combine knowledge acquired from several individuals into one expert system was introduced and evaluated by Boose (1986). In addition, a repertory grid-based knowledge acquisition tool, AQUINAS, an expanded version of the ETS, could be used to acquire knowledge from multiple sources (Boose, 1989). Another approach using a Group Decision Support System (GDSS) incorporating such techniques as brainstorming, analysis, and voting to facilitate the acquisition of knowledge from multiple experts was first proposed by Nunamaker et al. (1988). McGraw and Harbison-Briggs (1989) also suggested techniques such as brainstorming, consensus decision making, nominal group technique, and computer-facilitated sessions for working with multiple-expert teams.

The process of acquiring knowledge from multiple experts described in this paper is structured as a group activity. In the following sections, a GDSS environment is defined. A methodology for knowledge acquisition from multiple experts in such an environment is presented. Finally, the paper concludes with results of a field study in which this methodology was applied to develop an expert system for an information center.

2 A GROUP DECISION SUPPORT SYSTEM ENVIRONMENT

A GDSS is an integrated computer-based system that facilitates the solution of an unstructured or semi-structured task by a group of people (DeSanctis and Gallupe, 1987). A GDSS environment is more than a computer-based system. It includes not only hardware, software, facility, and people involved in the group process, but also procedures of particular group processes and group facilitation techniques. As used in this paper, the term GDSS

includes the concept of computer supported group work. Other meanings embraced by the term as used in this paper are computer-supported collaborative work, group support systems, and electronic meeting systems (Dennis et al., 1988).

There are many benefits to be derived from using a GDSS environment to extract knowledge from a group of experts. First, it appears that by bringing experts together in a computer-supported environment and by using group support tools to facilitate group interaction and information exchange, a knowledge engineer can significantly speed up the knowledge acquisition process. Brainstorming, voting, nominal group, and Delphi techniques are examples of some of the group methods that can be used (Huseman, 1973; Linstone and Turoff, 1975; Moore, 1987). Communication between the knowledge engineer and the participating experts and among experts themselves is very important. An effective GDSS environment is designed to increase efficiency and effectiveness of group activities and aims to improve the group process by removing common communication barriers and by directing the pattern, timing, or content of discussion. Additionally, the knowledge elicited can be documented immediately by electronic means. The process is also facilitated by the atmosphere of cooperation engendered by such an environment. Finally, the essential contribution of using a GDSS environment is that the overall quality and quantity of results from the group process appear to be superior to the sum of each individual's contribution.

2.1 Components of a GDSS Environment

A GDSS consists of hardware, software, people, and facility (DeSanctis and Gallupe, 1987). In addition, procedures (Dennis and his colleagues, 1988) and facilitation (Vogel et al., 1987), are also important part of a GDSS environment. All six are necessary to the knowledge acquisition task (Liou, 1989). The role of each component in the knowledge acquisition process and the interrelationships among them are depicted in Figure 1. As can be seen in the figure, the six components are represented by rectangles with rounded corners, whereas possible instances of each component are shown in rectangles. Thin lines marked with "include(s)" in shaded rectangles represent relationships between a component and possible instances while thick lines marked with verbs in shaded rectangles show relationships between two components.

The people component includes both participants and facilitators. In the context of knowledge acquisition, participants are domain experts and facilitators may be professionally trained facilitators or knowledge engineers who possess group technique competence and have experience in using various group support tools. The role of a facilitator during knowledge acquisition sessions can be one of three: group leader in terms of guiding the group to achieve objectives of particular sessions, chauffeur who serves as the intermediary between the group and the software, and assistant in the use of computer technology. Therefore, facilitators enforce

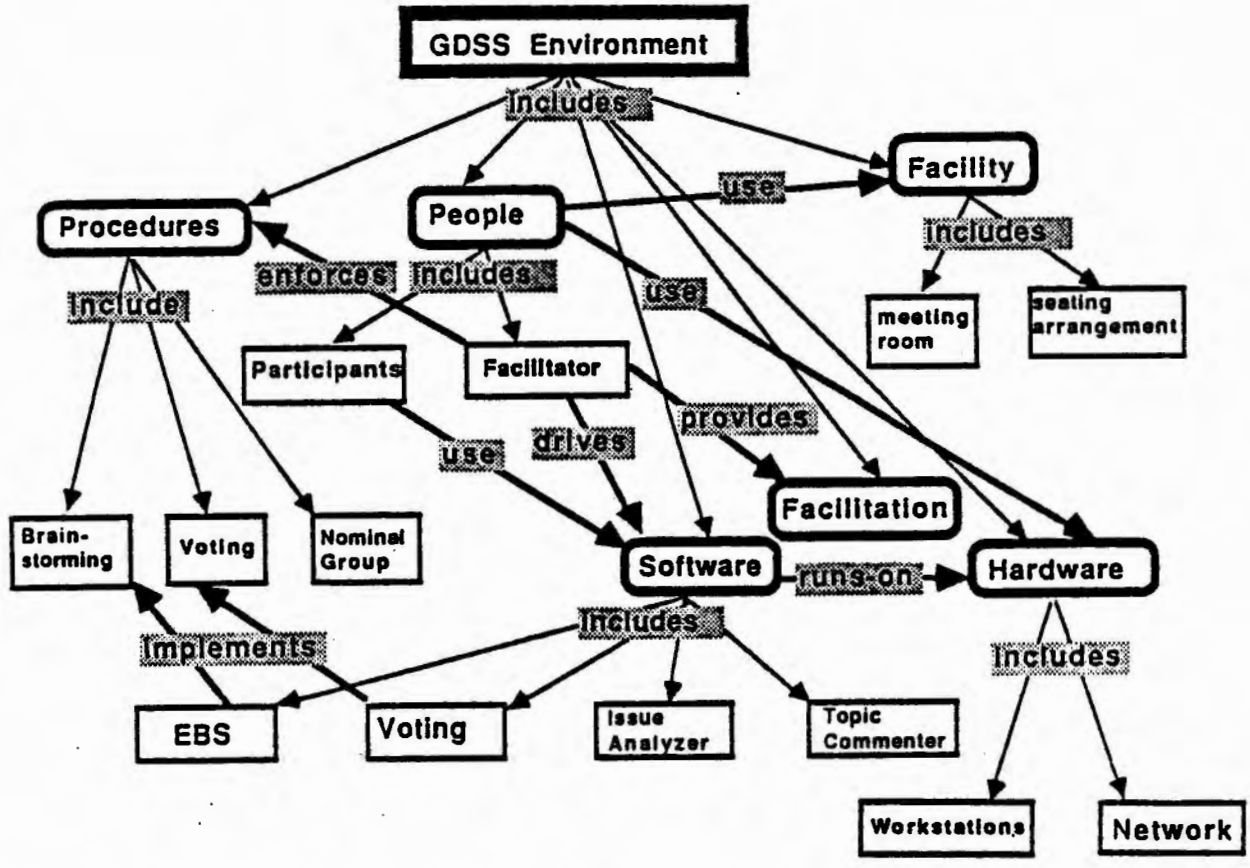


Figure 1 Interrelationships of Components of a GDSS Environment

procedures, assist participants, and initiate software programs. Both participants and facilitators use the facility.

In general, the facility comprises the environment in which group activities occur. Characteristics of the room such as observation windows, carpeting, adjustable lighting, seating arrangement appropriate to group activities, air conditioning, microphone, and noise control devices provide a setting within which people can comfortably work on complex problems. Procedures are particular group techniques that can be applied to the acquisition of knowledge. Brainstorming, nominal group, voting, and Delphi are some examples. Procedures are enforced by facilitators or knowledge engineers and are implemented by group support tools. Facilitation is the support provided by the facilitator who assists participants in using the computer technology, monitors group dynamics, and coordinates group sessions.

The hardware component includes individual workstations, central file servers, high-speed printers, presentation media, and a communication network to facilitate group communication. Individual workstations and a communication network allow participants to interact through the GDSS. Central file servers coordinate and

manage input from each individual and act as a knowledge repository from session to session. High-speed printers provide quality printouts with which participants can review the meeting proceedings. Presentation media provide a wide range of support that includes public and individual display of information, audio and video recording, electronic blackboards, and overhead projection systems.

The software component is the most important one among the six. It includes computer programs that comprise the overall system and group support tools designed to implement various group techniques, such as brainstorming, consensus decision making, and nominal group technique. Group support tools facilitate the processes of generating, analyzing, and consolidating concepts, ideas, objects, or class hierarchy into a structured format. Electronic Brainstorming, Issue Analysis, Topic Commenter, Voting, Nominal Group, and Delphi are examples of some group support tools developed for the PLEXSYS project at the University of Arizona by the Management Information Systems Department (Plexsys User Guide, 1988). What group support tools can provide in terms of functionality directly affects the outcomes of the group process. In the next section of the paper, a three-dimensional framework is used to analyze the characteristics of group support tools. A similar technique has been used to discuss visual programming languages (Shu, 1985).

2.2 Dimensional Analysis of Group Support Tools

Group support tools are the most important part of a GDSS environment. Typically, each tool is designed to implement a specific group technique and is appropriate for a particular set of tasks. They can be analyzed on the basis of: process capability, format of output, and interaction among participants.

o Process Capability. Group support tools fall along a continuum that ranges from analysis at one end to generation of concepts at the other. Analysis includes the classification, decomposition, and consolidation of ideas or concepts. Generation includes the divergent and convergent production of ideas or concepts and is stimulated by interaction among participants. Ideas produced by convergent production tend to have more relationship to one another than do those generated by divergent production.

o Format of Output. The format of output generated from the group process using a particular tool is the second dimension along which group support tools can be classified. This dimension ranges from totally unstructured or messy information such as outputs from brainstorming sessions to structured information such as outputs from voting sessions. Structured information is a like-order list of things. Unstructured information, for instance, can be of a hundred items of information generated by twenty people, some of whom address the same issue but in different ways, whereas other items may have nothing in common. It requires an enormous amount of effort to analyze and consolidate the output in order to make

it useful. A more structured form of output can consist of a hundred ideas from twenty people organized by them into ten categories. A great deal less effort is needed to analyze output in this form.

o Interaction among Participants. In a group-oriented task, interaction among participants can affect the generation and analysis of ideas. Tools can be designed or used to encourage interaction among participants and to allow them to respond to the comments of others in the group as well as to follow their personal trains of thought. Alternatively, tools can be designed to shield participants from the ideas of others so that they can either focus on their own points of view or work at their own rates of speed.

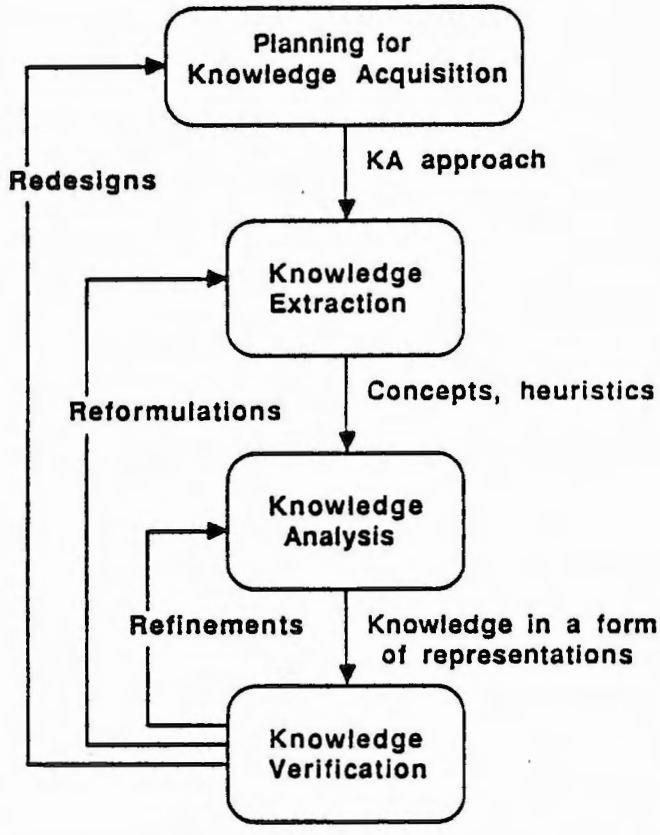


Figure 2 Knowledge Acquisition Phases

3 A METHODOLOGY FOR KNOWLEDGE ACQUISITION IN A GDSS ENVIRONMENT

The process of knowledge acquisition in a GDSS environment can be decomposed into four phases: planning for knowledge acquisition, knowledge extraction, knowledge analysis, and knowledge verification. These phases are depicted in Figure 2. The four stages comprise fourteen steps summarized in Table 1.

3.1 Planning for Knowledge Acquisition

In a GDSS environment, this is the most important phase of the knowledge acquisition task. The goal is to analyze approaches to knowledge acquisition and to design the procedures for acquiring required knowledge. Comparable to the analysis and design phase of the system development life cycle, it begins with the attempt to understand the application domain. This is followed by definition of the scope of the problem domain. Next is identification of the type of expert system applications; this enables knowledge engineers to determine the approaches they must take in acquiring domain knowledge. Group techniques that can be used to extract knowledge are subsequently identified and appropriate group support tools are selected. Process models that integrate group techniques

KA Phases	Steps
Planning for KA	Understanding the domain Defining the problem scope Identifying the type of Application Developing process models Identifying participants Planning knowledge acquisition sessions
Knowledge Extraction	Explaining the KA Discussing the objectives of KA sessions Conducting KA sessions Debriefing the expert team
Knowledge Analysis	Analyzing sessions outputs Transforming knowledge into representations
Knowledge Verification	Developing test scenarios Verifying knowledge with a panel of experts

Table 1 Knowledge Acquisition Phases and Steps

and tools are then developed. Participants in knowledge acquisition sessions must be identified. Finally, a detailed plan of the ways that the knowledge acquisition sessions are to be conducted must be developed. When planning has been completed, the knowledge acquisition approach and the procedures for conducting the various sessions are identified.

Understanding the Domain. The initial phase of an expert system development project has a great impact on the effectiveness of the entire process. No matter what methods or techniques a knowledge engineer uses to acquire knowledge, an understanding of the terminologies, concepts, and problem-solving strategies of the application domain is necessary. As Prerau (1987) noted, knowledge engineers are trained to "speak the language" and understand the basic concepts in the domain. The goal is not to turn knowledge engineers into experts but to increase the effectiveness with which they communicate with experts and conceptualize the domain. Ways to help knowledge engineers understand the domain include: self-study, development of a knowledge dictionary, attendance at seminars or lectures, and establishment a reference center (McGraw and Harbison-Briggs, 1989).

Defining the Problem Scope. The success of an expert system is closely related to the scope of the problem domain. Davis (1984) noted that expert systems deal with very narrow domains of expertise. If the scope is too broad, the system can not be effective because too much knowledge must be acquired, managed, and stored. In addition, it is difficult to identify experts with the required expertise. When the problem scope is clearly stated and focused on a specific area or on a few related areas, domain experts can be more easily identified.

Decisions about problem scope can be made by domain experts, end users, managers, and the knowledge engineer or any combination of these individuals. The involvement of domain experts in the decision provides a narrow perspective of what the system can do for users, while the involvement of end users provides a broader understanding of how the system may be able to help them. Managers are involved in order to provide strategic guidelines as to what should and should not be included in the system. The knowledge engineer can assess technical feasibility. One individual can, of course, determine scope, for example, the manager could do so on the basis of organizational policy and requirements. In most cases, however, users should be involved because decisions that they have helped make are generally more acceptable to them.

Identifying the Type of Application. Problems that expert systems are capable of solving can be grouped into two categories: analysis and synthesis (Clancey, 1986). EMYCIN, a medical consultation system, is an example of a system that addresses a typical analysis problem (Hayes-Roth, 1983). XCON, a configuration system for VAX computers, deals with synthesis type problems (Waterman, 1986). Each category has its own characteristics and requires different problem-solving approaches. Expert systems that deal with analytic problems look, in general, for solutions in the domain as a whole and try to narrow the domain, with the search space getting smaller and smaller until solutions are identified. A top-down approach is appropriate for problems in this category because it requires that experts decompose a problem into subproblems and tackle smaller problems first.

Problems of the synthesis type have different requirements. First, all necessary information must be identified in detail. This information is then combined pursuant to synthesis rules. For example, to configure a car, a worker first identifies all the parts required and then puts those parts together. Different combinations of different types of engines, bodies, and tires results in various car models. A bottom-up approach seems suited to solving problems in this category. It requires that experts first generate all the necessary subproblems; these are then composed into ever larger problem components.

Developing Process Models. To maximize the benefits of using a GDSS environment in the acquisition process, knowledge engineers must select tools that best fit the application type so the tools support the process in the most effective manner possible. This requires an analysis of the experts' problem-solving tasks. Through task analysis, knowledge engineers are not only able to define major problem-solving tasks but they are also able to determine what types of problems the system will be required to handle and the types of information it will need in order to provide solutions to those problems.

After task analysis has been completed, the knowledge engineer needs to design a series of actions that perform these tasks. These

actions can be supported by computer-based systems, by group-process techniques, or by a combination of both. In this paper, the term "process model" is defined as a designed series of actions that bring about desired results. In the context of knowledge acquisition, actions are the use of various tools to acquire knowledge, and the result is acquired knowledge. Five process models, session planning, identification, classification, repertory grid analysis, and verification, have been developed using the current PLEXSYS Tool Kit (Liou, 1989). The classification model, for instance, depicted in Figure 3, is an iterative process which develops a classification scheme as sessions proceed. It involves four steps: session planning, parallel brainstorming, identifying issues, and consolidating issues. Detailed description of these models are provided in (Liou, 1989). Whatever the domain, in the future as needs for new group process support tools are identified, it will most likely be necessary to identify domain-specific tasks and task sequences and to generate new domain-specific process models. The activities identified in the model will then need to be linked to support mechanisms. In a GDSS environment like the one described in this paper, those activities or tasks that require computer support can be fairly easily implemented.

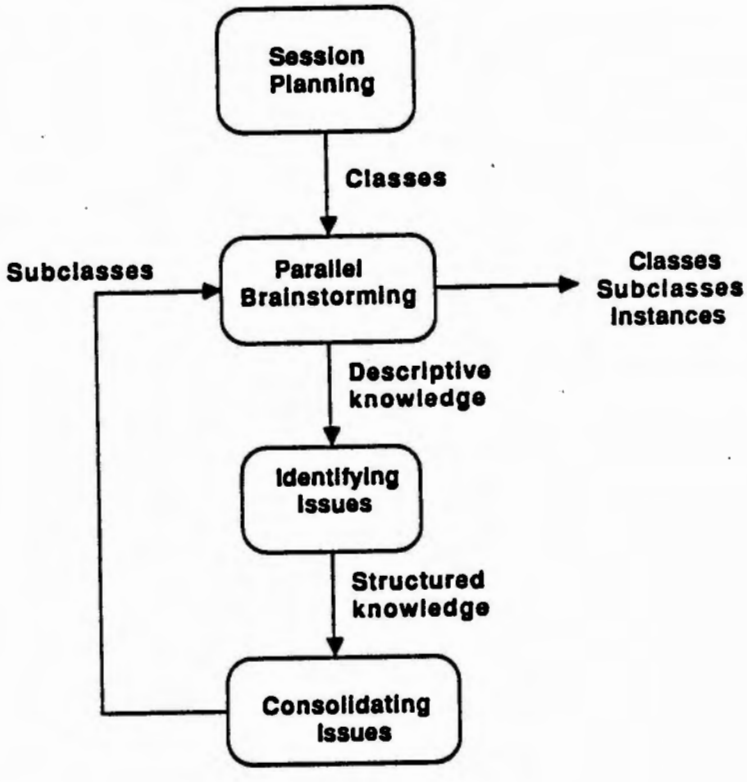


Figure 3 Classification

In designing and developing process models, knowledge engineers must know for what group techniques the current tool kit provides and the tasks it supports. This is achieved through dimensional analysis. Second, they must have had enough experience in working with the group support tools to know their strengths and weaknesses as well as what can be done with them. Third, knowledge engineers must understand the problem domain, the problem-solving process, and the tasks involved in that process in order to match tools with the required knowledge acquisition tasks.

Identifying Participants. For the most part, participants in

knowledge acquisition sessions are limited to experts and facilitators. Users and managers are usually involved only in defining the problem scope. Consequently, identification of experts is fundamental to success. This task is a joint responsibility of organizational managers and knowledge engineers. By analyzing the domain and problem characteristics, it is possible to pinpoint sources of expertise. Attributes that should be considered when choosing domain experts include domain background, customer authorization, availability, personal characteristics, and attitudes (McGraw and Harbison-Briggs, 1989).

Domain background includes the experiences an individual has had and the capacity in which the person has used this knowledge. It is the practicing experts who need to be identified not the experienced experts who are not currently practicing in that domain (McGraw and Seale, 1987). The term "customer authorization" refers to the issue of who decides whether an individual is an "expert." Because personal characteristics and attitudes may influence the transfer of information to the knowledge engineer, such factors as the expert's sense of humor, ability to send as well as receive information, sense of commitment, patience, metaknowledge, willingness to participate, honesty, and persistence (McGraw and Harbison-Briggs, 1989) should be taken into consideration. Limited access to domain experts is a common problem in knowledge acquisition. Hart (1985) noted that most problems are caused by time demands on an "already-busy" expert. Regardless of their experience and characteristics, individuals who are overloaded should not be selected as primary experts. One solution to the access problem is to gain management support for the project so that experts are given the time to participate.

Identification of the facilitator is also extremely important. Facilitators must be technically competent. They must be able to use GDSS software, direct the system, control environmental factors like lighting, and establish effective procedures. The ability to work well with others is also important. This includes communication skills and mastery of group facilitation techniques. During knowledge acquisition sessions, professional facilitators are desirable, but knowledge engineers with special training in group facilitation techniques may be acceptable.

Planning Knowledge Acquisition Sessions. Preparation for knowledge acquisition sessions is the final step in the planning stage. Objectives of each group session must be identified. A session agenda describing objectives, participants, duration, input, processes, expected output, and the overall sequences of events must be developed. Intermediate activities between sessions must be identified. These may include backing up and reviewing session outputs and preparing files for subsequent sessions. Input files such as brainstorming issues, discussion topics, and background information should be prepared before sessions start. In addition, notices of sessions must be delivered to participants and reservations for the GDSS facility must be made.

3.2 Knowledge Extraction

When planning is completed, the knowledge extraction phase starts. The primary activity during this phase is to acquire knowledge from experts through a series of knowledge acquisition sessions. It may consist of various sets of sessions, each with a particular objective and each using specific knowledge acquisition techniques in conjunction with group support tools. During each session, knowledge engineers must explain the session objective, the techniques to be used, and the expected results. Functionalities of the group support tools and the techniques of using them are explained. Outputs from this phase include heuristics, concepts, and classification structures. These may be captured in an electronic format and require further analysis to be represented in particular schemes.

Explaining the Knowledge Acquisition Approach. The knowledge extraction phase must begin with an explanation of knowledge acquisition so that participants acquire an overall picture of the process. Basic concepts about it and its role in the expert systems development life cycle must be explained. This should be done at an introductory rather than an in-depth level. In addition, the specific approach taken with respect to the development of the target system must be discussed. For instance, if help-service problems in an IC are to be decomposed, a discussion of the classification approach should point out that an expected output from sessions is a taxonomy of help-service problems. These explanations establish expectations for experts as to what they need to contribute in forthcoming sessions, and, thereby, facilitate the acquisition process.

Discussing the Objectives of Knowledge Acquisition Sessions. Objectives of sessions must be identified during the planning stage. An overview of the objectives of the entire series of knowledge acquisition sessions and the objectives of individual sessions should be presented and discussed in order to reduce potential misinterpretations during the process.

Conducting Knowledge Acquisition Sessions. The use of structured procedures affects the result of these sessions. A structured procedure applies such techniques as brainstorming and voting to acquire knowledge from a group of experts, organizes sessions according to particular process models, and integrates inputs and outputs from session to session. A contingency plan and frequent backup of session outputs are recommended in case of a system failure or any other unexpected problems.

Facilitation, a second factor that significantly affects session results, requires technical competence, communication skills, domain knowledge, and group facilitation skills. Technical competence includes knowledge of the group support tools: how to use them, their purposes and limitations, and how they work together to support the knowledge acquisition process. Knowledge of the problem domain is task-oriented. Group facilitation skills

require a sound understanding of group dynamics and possession of the skills to work within the constraints of those dynamics to help the group achieve goals. Facilitators who are not knowledge engineers must be aware of the expectations of the knowledge engineers who plan and organize the sessions and of the experts who take part. Having a facilitator who fail to take these different expectations into account are likely to fail.

Debriefing the Expert Team. Debriefing is the last step of the knowledge extraction phase and it brings closure to the knowledge acquisition sessions. It is especially important when sessions have involved a group of experts. Debriefing should address the tasks of summarizing and correctly recording the problems, solutions, and ideas raised by participants. Experts should be encouraged to request or suggest clarification of the issues that were raised. The debriefing process offers knowledge engineers an opportunity not only to obtain consensus from experts, if desired, but also to elicit data concerning the experts' degree of certainty or belief in the information obtained. While certainty or validity of rules for a knowledge base is always important, it is extremely so in situations in which more than one expert may have contributed ideas or in which ideas were consolidated based on discussion among experts (McGraw and Seale, 1987; 1988). As the session concludes, the knowledge engineer should take an opportunity to identify action items, parties responsible for specific tasks, and target completion dates. Additional objectives for debriefing the expert team include ensuring that the group members have positive attitudes about their contributions and their participation and outlining future plans for the team.

3.3 Knowledge Analysis

The primary task of this phase is to analyze outputs from knowledge acquisition sessions. Heuristics and concepts are analyzed and formalized into representations that may be in the form of rules, frames, objects and relations, semantic networks, and classification schemes. These representations are then transferred into particular representation schemes that are supported by the expert system building tool (i.e., expert system shell). Although implementation of the expert system is not an issue in knowledge acquisition, it determines what knowledge representation schemes can be used. Thus, knowledge engineers must decide how an expert system is to be implemented in terms of selection of expert system building tools, knowledge representation schemes, and inference methods before the analysis phase begins.

Analyzing Sessions Outputs. Outputs from knowledge acquisition sessions must be analyzed and consolidated into knowledge chunks, which in turn must be formalized into such representations as classes and instances, semantic networks, or a hierarchical classification tree. Certain guidelines should be followed during analysis. These are illustrated in the following animal classification system:

o Identify concepts, objects, or entities; most of the time, present them as nouns, for example, mammals, birds, dogs, and cats.

o Identify attributes associated with each concept, object, or entity; most of the time, present them as adjectives or descriptors that can be used to differentiate one concept or entity from another. These indicate object-attribute relationships. For example, mammals drink milk; birds fly; dogs bark.

o Identify possible values of attributes; most of the time present them as numbers, adjectives, or descriptors. For example, human beings have two arms and two legs; dogs have four legs; cats have four legs; the color of a horse may be white, brown, or black.

o Identify class-instance relationships. Verbs such as "is a," "a kind of" imply such a relationships. For example, a dog is a kind of mammal. In other words, dogs are a subclass of mammal.

o Identify part-subpart relationships. Verbs such as "has parts," "has a," "consists of," "is part of" imply such relationships. For example, a head consists of eyes, ears, nose, mouth, and hair.

o Identify heuristic rules using terms such as "if...then..." as indicators. For example, if an animal drinks milk, then it is a mammal.

A parsing program may be used to assist in the identification of knowledge items such as concepts, objects, entities, attributes, values, relationships, and heuristic rules. Visual diagrams such as semantic networks, hierarchical classification structures, and object-attribute-value frames help represent knowledge in a particular format.

Transforming Knowledge into Representations. Knowledge items like objects, attributes, values, relationships, and heuristic rules resulting from the previous step are transformed into representations supported by the expert system building tool. For example, the help-service system was developed on the Expert System Environment (ESE/VM), a rule-based expert system shell. Some guidelines for transforming knowledge items into rules include:

o Use a diagram to represent knowledge chunks. Such diagrams may be a hierarchical classification scheme, a semantic network, or object-attribute-value frames.

o Transfer object-attribute relationships into rules. For example, it may be a fact that birds fly. A heuristic rule derived from this fact would be: If an animal flies, then it is a bird.

o Transfer class-instance relationships into rules. For example, it is a fact is that dogs are mammals. The associated rule would be: If Snoopy is a dog, then Snoopy is a mammal.

o Transfer part-subpart relationships into rules. For example, a dog has a head, a body, and legs. The heuristic rule would be: If Snoopy is a dog, then Snoopy has a head, a body, and legs.

The more experience or expertise a knowledge engineer has in building expert systems, especially in representing knowledge extracted from human experts, the more reliable and effective the resulting expert system. Unfortunately, analysis and transformation of knowledge chunks into representations are still performed manually. A computer program to assist knowledge engineers in transforming knowledge chunks into formal knowledge representations would contribute significantly to their productivity.

3.4 Knowledge Verification

This phase focuses on verifying heuristics, concepts, and classification structures. Formalized representations are discussed with the group of experts. Demonstrations of the prototype system also helpful. Represented knowledge can be refined by looping back to the knowledge analysis phase, whereas reformulation of heuristics, concepts, or classification structures must be accomplished by returning to the knowledge extraction phase. If knowledge captured in the prototype system does not provide solutions to problems, redesign of the knowledge acquisition sessions is necessary. This requires rethinking the knowledge acquisition approach as well as procedures to acquire knowledge.

Developing Test Scenarios. The first step in the knowledge verification phase is to develop test scenarios. Potential system users are the best source available. Knowledge engineers should sample problems that users commonly encounter and use them to test the knowledge base. Engineers may also create various scenarios based on their own understanding of the problem domain. However, scenarios collected from users are generally more realistic and, therefore, better tests of the system's capabilities.

Verifying Knowledge with a Panel of Experts. There are three levels of verification that must be considered. First-level verification involves providing session outputs to experts for comments. A limitation of this level is that experts may focus on wording or phrasing rather than on the completeness of the knowledge elicited. The engineer must help them think about the extent of what is to be included in the knowledge base rather than on assorted items of information. Verification results are used to refine the knowledge base. These refinements are analyzed and transformed into representations which are then incorporated into the existing knowledge base.

Second-level verification involves explaining knowledge structures to the group of experts. This helps them get a rough idea of what will be included in the knowledge base so that missing pieces of knowledge can be recognized and submitted for refinement.

Third-level verification involves using the prototype system to test the scenarios that have been developed. Solutions to testing problems provided by the prototype system are verified by the panel of experts, some of whom have been involved in knowledge acquisition sessions and some of whom have not. Additionally, experts may elect to use their own scenarios to test the system.

4 A TEST OF THE METHODOLOGY

The knowledge acquisition process described in this paper was tested by means of a field study. Specifically, a GDSS environment was used to develop an expert system for an IC in a large corporation (Liou, 1989). By using the methodology presented in the paper, it required only four and a half days to elicit the knowledge required for the IC's Help-Service System from the center's consultants. Planning took one day. Knowledge extraction took two days. Verification required one and a half days. Preliminary feedback about the current implementation of the system has been quite positive.

4.1 Electronic Documentation of Knowledge

One of the primary advantages of using computer support to facilitate the acquisition of knowledge from a group of experts is that knowledge is documented electronically by the computer. There is no need to tape record the interview process so that time required to transcribe audio or video tapes is saved. Although the format of outputs from knowledge acquisition sessions may be unstructured, they are captured while sessions are conducted. Experience gained in this field study shows that videotaping during sessions in which oral discussion is the primary communication channel is necessary, since oral conversations cannot be captured by current GDSS software. These videotapes can also be used to study such behavioral processes as conflict resolution.

4.2 Parallel Knowledge Extraction

When knowledge engineers elicit knowledge from individuals and then integrate the knowledge of each expert, the process is typically time-consuming and difficult. It is time-consuming because conflicts in problem-solving methods cannot be identified during the extraction process and must be resolved through several iterations of individual knowledge elicitation. It is difficult because different engineers are not equally capable. Consequently, results from individual knowledge elicitation sessions vary. Another important consideration is that integration of individual knowledge into group knowledge is more properly performed by experts than by knowledge engineers.

One of the advantages anticipated in the field study was that concurrent (or parallel) acquisition of knowledge from experts would eliminate these difficulties, and, indeed, use of a GDSS environment was both efficient and effective. Because it was a parallel process, the knowledge engineer was able to take advantage of such techniques as group interviewing, group discussion, and brainstorming. Additionally, duplications and conflicts were addressed as elicitation occurred. This saves time in both the knowledge extraction and analysis phases.

4.3 Conflicts Addressed during Knowledge Extraction

Because a GDSS environment was used, conflicting viewpoints were addressed during the knowledge extraction phase. Conflicts arose mainly when experts' heuristics were used to classify common problem areas and identify problem instances and solutions to those instances during the extraction process. The tools, Electronic Brainstorming (EBS) and Topic Commenter, allowed each expert to see and comment on the input of others in the group during knowledge acquisition sessions. Discussions were encouraged at all times even when electronic media were not used. Results of the field study suggest that discussions, whether in face-to-face or distributed environments, should be encouraged throughout the process.

4.4 Interaction Resulting in Enlarged and Enriched Domain of Expertise

Even though they were experts, those involved in the GDSS experience learned more about their domains of expertise. For instance, several brought up problems others had not previously encountered as well as solutions to those problems. On other times, experts discussed nuances in their individual responses to common problems. Furthermore, experts' comments frequently set off productive and interesting chains of association in the thinking of others in the group. Experience gained from this field study suggests that tools must encourage interactions among participants by providing a variety of communication channels, either through the system or independent of it. It is preferable to use the system, since discussions can be captured by the system so that videotaping is not necessary.

4.5 Planning for Knowledge Acquisition using Structured Analysis Techniques

Structured analysis techniques were used to help structure the domain and identify knowledge acquisition needs. These techniques included task and job analysis (McGraw and Harbison-Briggs, 1989). Because these analyses served to identify and delimit major sections of the domain, they provided development focus and facilitated planning. The analyses identified the major functions or tasks for which the expert system was to be responsible. It helped the knowledge engineer identify specific areas for knowledge acquisition sessions, thereby reducing the amount of time required

for knowledge acquisition and making it easier to plan or schedule knowledge acquisition for the prototyping cycle. Finally, because goals had been clearly specified, the sessions were fairly easy to manage.

4.6 A Designated Primary Expert

Having the right person in the role of primary expert is critical to the success of the project. In the field study, this individual was of tremendous help to the knowledge engineer in identifying other experts, scheduling knowledge acquisition sessions, gathering experts, explaining the objectives of sessions to other participants, and helping resolve conflicts and refine knowledge. This finding supports Prerau's (1985) and Sviokla's (1986) recommendation that a primary expert be identified when a team approach is used.

5 CONCLUSIONS

The results of this field study indicate that a GDSS environment does facilitate the acquisition of knowledge from multiple experts. There are six major advantages in using a computer-supported cooperative approach. First, knowledge is documented electronically. Second, knowledge extraction from individual experts can be performed in parallel. Third, conflicts can be addressed during the knowledge extraction phase. Fourth, interaction among experts results in an enlarged and enriched domain of expertise. Fifth, structured analysis techniques such as task analysis can be used to plan for knowledge acquisition. Finally, a designated primary expert can be of great help when dealing with multiple experts. Future studies to apply this methodology in acquiring knowledge in various domains and for different types of expert-system applications should be equally instructive.

ACKNOWLEDGEMENTS

This research was supported in part by IBM.

REFERENCES

Boose, John H. (1986). "Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain," Future Computing Systems, Vol. 1, No. 2, pp. 191-216.

Boose, John H. (1989). "Using Repertory Grid-Centered Knowledge Acquisition Tools for Decision Support," Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, Vol. III, Kona, HI, January 3-6, pp. 211-220.

Chorafas, D.N. (1987). Applying Expert Systems in Business, McGraw-Hill.

Clancey, William J. (1985). "Heuristic Classification,"

Artificial Intelligence, Vol. 27, pp. 289-350.

Davis, Randall. (1984). "Amplifying Expertise with Expert Systems," The AI Business: Commercial Uses of Artificial Intelligence, edited by P.H. Whinston and K.A. Prendergast, The MIT Press, Cambridge, MA, pp. 17-40.

Davis, Randall. (1979). "Interactive Transfer of Expertise: Acquisition of New Inference Rules," Artificial Intelligence, Vol. 12, pp. 121-157.

Dennis, Alan R., George, J.F., Jessup, L.M., Nunamaker, J.F., and Vogel, D.R. (1988). "Information Technology to Support Electronic Meetings," MIS Quarterly, Vol. 12, No. 4, December, pp. 591-624.

DeSanctis, Gerardine and Gallupe, B. (1987). "A Foundation for the Study of Group Decision Support Systems," Management Science, Vol. 33, May, pp. 586-609.

Feigenbaum, E.A. (1977). "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," International Joint Conference on Artificial Intelligence 5, pp. 1014-1029.

Hart, Anna. (1985). "Knowledge Elicitation: Issues and Methods," Computer-aided Design, Vol. 17, No. 9, pp. 455-462.

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

Huseman, R. (1973). "The Role of the Nominal Group in Small Group Communication," in R.C. Huseman, D.M. Logus, and D.L. Freshley, eds., Readings in Interpersonal and Organizational Communication, 2nd ed., Hollbrook, Boston, MA.

LeClair, S.R. (1985). A Multiexpert Knowledge System Architecture for Manufacturing Decision Analysis, Doctoral dissertation, Arizona State University, May.

Linstone, H. and Turoff, M. (1975). The Delphi Method: Techniques and Applications, Addison-Wesley, Reading, MA.

Liou, Yihwa Irene. (1989). The Use of a Group Decision Support System Environment for Knowledge Acquisition, Ph. D. dissertation, The University of Arizona.

McGraw, K.L. and Harbison-Briggs, K. (1989). Knowledge Acquisition: Principles and Guidelines, Prentice Hall, Englewood Cliffs, NJ.

McGraw, Karen L. and Seale, M. (1988). "Knowledge Elicitation with Multiple Experts: Considerations and Techniques," Artificial Intelligence Review, Vol. 2, No. 1, January/February.

McGraw, Karen L. and Seale, M. (1987). "Multiple Expert Knowledge Acquisition Methodology: MEKAM," Proceedings of the Third Australian Conference on Applications of Expert Systems, The New South Wales Institute of Technology, Sydney, May, pp. 165-197.

Mittal, S., and Dym, C.L. (1985). "Knowledge Acquisition from Multiple Experts," The AI Magazine, Vol. 6, Summer, pp. 32-36.

Moore, C.M. (1987). Group Techniques for Idea Building, Sage Publications, Newbury Park, CA.

Nunamaker, J.F., Konsynski, B.R., Liou, Y.I., Chen, M., Vinze, A., and Heltne, M. (1988). "Use of a GDSS for Knowledge Acquisition in Information Center Expert," Working paper, University of Arizona, Department of Management Information Systems, October.

Plexsys User Guide. (1988). Department of Management of Information Systems, University of Arizona.

Prerau, D.S. (1987). "Knowledge Acquisition in the Development of a Large Expert System," The AI Magazine, Summer, pp. 43-51.

Prerau, D.S. (1985). "Selection of an Appropriate Domain for an Expert System," The AI Magazine, Vol. 6, Summer, pp. 26-30.

Shu, Nan C. (1985). "Visual Programming Languages: A Dimensional Analysis," Proceedings of IEEE International Symposium on New Directions in Computing, August, pp. 326-334.

Sviokla, J.J. (1986). "Business Implications of Knowledge-Based Systems," Parts I and II, Data Base, Vol. 17, Summer, pp. 5-19, Vol. 18, Fall, pp. 5-16.

Vogel, Doug, Nunamaker, J.F., Applegate, L., and Konsynski, B. (1987). "Group Decision Support Systems: Determinants of Success," DSS-87 Transactions, O.A. El Sawy ed., pp. 118-128.

Waterman, D.A. (1986). A Guide to Expert Systems, Addison-Wesley, New York.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Developing An Integrated Toolkit for Cognitively-Based Knowledge Acquisition

Karen L. McGraw
Cognitive Technologies
2315-B Forest Dr., #9
Annapolis, MD 21401

Juliana S. Lancaster
The BDM Corporation
1300 N. 17th, Suite 950
Arlington, VA 22209

INTRODUCTION

As increasing numbers of knowledge-based systems are developed for operational use, the importance of knowledge acquisition becomes even more evident. Knowledge acquisition, the process of extracting and transferring expertise from domain experts into the system, consumes the single largest block of development time (Feigenbaum, 1977; Hayes-Roth, et al., 1983; McGraw & Harbison-Briggs, 1989). To reduce the time required for this process, numerous tools have been developed that automate or systematize the elicitation and transfer of knowledge from expert to knowledge base. Among these advances are developments in machine learning that enable an expert system to acquire its knowledge from "experience" (Michalski, 1980; Michie, 1982), expert system shells that allow domain experts to write their own rules (e.g., MacSMARTS), and knowledge engineering tools that assist a knowledge engineer in his/her work. The proposed paper describes research and development efforts on a computer-based set of tools known collectively as the BDM Knowledge Acquisition Toolkit (BDM-KAT).[1]

BDM-KAT has been designed and implemented on a SUN 3 workstation to aid a knowledge engineer in eliciting and representing expert knowledge. The design philosophy behind BDM-KAT is based on an application of the theories of Dr. L. M. Vekker (1974, 1976, 1981) concerning the representation and structure of knowledge in memory.[2] A critical component of this set of theories is the belief that expert domain knowledge is stable across both time and perspective. The toolkit provides an environment in which the knowledge engineer and domain expert discuss, manipulate, and record the objects and relationships that provide the domain with its structure and "predictability." As elicitation progresses, a database of basic domain components is developed. Using this database, a knowledge base and inferencing structure can be built to operate in the selected domain.

BDM-KAT differs from other tools in several critical ways. It has been designed and implemented to aid a trained knowledge engineer in eliciting expert knowledge about well-structured domains. It is based on principles of human cognition relating to the representation and structure of knowledge in memory, yet these representations are implemented in a computationally-efficient way. The toolkit provides an environment in which the knowledge engineer and domain expert discuss, manipulate, and record the objects and relationships that provide the domain with its structure and predictability. As the elicitation progresses, a database of basic domain components is developed. Using this database, a knowledge base and inferencing structure can be derived to operate in the selected domain. It is important to note that BDM-KAT can elicit rules about the domain and its operation, but that it is not required to do so.

---

[1]Work described in this paper was supported by the US Defense Advanced Research Projects Agency, through the Air Force Office of Scientific Research under contract number F49620-86-0036 and the Army Research Office under contract number DAAL03-87-C-0019. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency or the US Government.

[2]Dr. Lev Vekker is an eminent Soviet psychologist who has recently emigrated to the United States. He and a colleague, Dr. Edward Manukian, have been pivotal in refining the application of cognitive theory to BDM-KAT.

The development of BDM-KAT can be discussed from several perspectives, including its conceptual basis, representation, integration, and actual implementation. The purpose of this paper is to describe the application of cognitive theory in memory and knowledge organization to the structure and design of BDM-KAT. First, we discuss the importance of establishing valid conceptual frameworks during the knowledge acquisition process. Next, we outline the specific organizational structures that are reflected in the existing components of the toolkit. Finally, we describe the overall design, integration, and implementation of the tools that comprise BDM-KAT: Categorizer, Time Mapper, and Causal Charter.

A STATEMENT OF NEED

The field of knowledge engineering has been both fueled and stressed by the development of increasing numbers of knowledge-based systems. To develop these systems practitioners must use a methodology and techniques that ensure that the system be well-organized, consistent, and reasonably complete. Early efforts in knowledge acquisition and engineering were modeled on small scale projects in which the domain expert and a single developer interacted to build the final project. The project size allowed developers to approach the task informally and select knowledge acquisition techniques in an ad hoc manner. Current systems tend to be larger and the ad hoc nature of early knowledge engineering methodologies have proven to be unsuitable.

Early methodologies for knowledge engineering were often based on the five-step cycle of identification, conceptualization, formalization, implementation, and testing (Hayes-Roth, et.al., 1983). While this tact provided a structure within which a system architecture and basic domain structure could be defined, it does not provide any control or guidance on the knowledge acquisition techniques that were used or the techniques by which the system's content could be deemed "valid." More current methodologies tend to depict the knowledge engineering cycle in a more managed cyclical manner, drawing on approaches popularized in the software engineering field. Some methodologies, like the adapted waterfall model for knowledge acquisition (McGraw and Harbison-Briggs, 1989) suggest not only steps within the cycle, but techniques appropriate for each phase and guidelines for decision points and iteration. However, even this approach does little to address how the knowledge acquisition task might be tied directly to the perceived structures and representations within the domain, as viewed by an expert in that domain.

Within any manual knowledge engineering activity, the time and labor demands are high, even when attempts are made to structure and manage knowledge acquisition sessions. In addition, several characteristics of the process remain troublesome. Specifically, true experts, because of the demand for their expertise, are frequently unavailable for series of multiple knowledge acquisition sessions. When those sessions are held, knowledge engineers frequently experience communication problems that diminish the sessions' success. For example, experts, who have chunked knowledge specifics often have great difficulty accessing, and then verbalizing that knowledge. This leads, in turn, to the knowledge engineer's inability to tap and be able to depict the structure and relationships the expert holds of the domain. Furthermore, domain experts' expertise makes them unable to recognize what is important and leads them to assume the knowledge engineer's understanding, even when the assumption is unfounded. Finally, many knowledge engineers cling to the interview as the primary knowledge acquisition technique. In the hands of a skilled communicator with adequate analytical skills and a good understanding of the domain, this technique is useful. However, few knowledge engineers are able to use it efficiently, leading to a large expenditure of time resulting in few critical pieces of knowledge (and often, a disgruntled expert, as well). These problems are compounded by the knowledge engineer's tendency to establish system design and architecture efforts prior to extensive knowledge acquisition. This focus forces a way of thinking about the domain and the subdomains within it in a way that may not be parallel to that held by the domain expert. Finally, regardless of the techniques that are used, many manual-based knowledge acquisition efforts fail to adequately document knowledge in such a way that enables later verification and validation of the system.

KNOWLEDGE ACQUISITION TOOLING

A number of approaches to automating the process of knowledge acquisition to alleviate some of these deficiencies have been attempted. These automation efforts ranged from the use of simple problem simulations to fully-automated knowledge engineering environments. Most of these tools were developed to serve the needs of a specific system, a class of applications, or the program in which they were to be used. Thus, they are implementation-driven. Because they are oriented toward the final representation and the way that the resulting system was to use knowledge, these tools may not produce optimum results. For example, they constrain a domain expert's communication of his or her knowledge by requiring that he or she present knowledge in the format on which the tool is based or in the representation of the knowledge base (e.g., rules, frames). This requirement may force the experts to transform or translate their knowledge from a familiar, useful form to a novel, untested one. In contrast, many manual approaches allow the expert to express knowledge in a more comfortable manner, but the burden of transforming the knowledge rests on the knowledge engineer.

In designing BDM-KAT, we considered the characteristics that we desired in a knowledge engineering environment and set these as design goals. Specific design goals included the following:

- The tool should reduce the requirements for translations of knowledge, particularly by the domain expert
- The tool should reduce the burdens in time and effort on the domain expert
- The tool should reduce the dependence on the skill of the individual knowledge engineer
- The tool should provide a mechanism for selecting appropriate techniques
- The tool should provide an overall control structure for a knowledge elicitation effort.

To meet these goals we decided to base the system on a theoretical foundation from cognitive psychology and to control knowledge elicitation through the use of recognized conceptual mechanisms. The mechanisms that were selected included categorical structures, as outlined by Rosche, Mervis, Gray, Johnson, and Boyes-Braem (1975), causal models, and event timelines (Barsalou, 1986). The following section describes details of the cognitive framework on which BDM-KAT was based and the characteristics of human knowledge organization and reasoning that have been central to its design.

A MULTI-DIMENSIONAL MODEL OF HUMAN KNOWLEDGE

Two primary premises have driven BDM-KAT's design. First, all aspects of an expert's knowledge must be captured in a knowledge based that can be used by an intelligent system. Equally important is the premise that the knowledge structures as elicited and represented must be compatible with the structures used by the expert. There is substantial agreement with both of these premises within the AI and cognitive science communities (Barfield, 1986; Gammack, 1987; Hamill, 1984; Kidd, 1983; McGraw and Harbison-Briggs, 1989). However, the problem of eliciting and representing non-verbal knowledge is an extremely difficult one, particularly because the nature of non-verbal knowledge is not well-understood by cognitive psychologists. Generally speaking, the cognitive aspects of non-verbal knowledge have been treated in more detail by Eastern psychologists than those of North America. In the States controversy persists about the reality of direct visual representation.[3] The work of Professor

[3]This disagreement is represented by the advocacy of cual coding by Paivio and its rejection by Pylyshyn (1986).

Leo M. Vekker (1974, 1976, 1981) stands as a well-thought-out example of Eastern European theories. Vekker's theory of human knowledge processing integrates all levels of mental activity from perception to abstract thought. In the following paragraphs we provide a brief summary of the aspects of this theory that have been central to the design of BDM-KAT.

Over a research career of 40+ years Vekker developed a comprehensive theory of the human cognitive system. To develop a multi-dimensional model of the human cognitive system, he drew strongly on the general structure of information theory. Specifically, he defines knowledge as a special type of information within that framework and uses the principle of isomorphism to draw parallels between information theory and levels and types of knowledge within the human system. Vekker describes the internal store of information as representing two different types: imagistic and sentential knowledge. These, in turn, are reflected in multiple levels of the cognitive system. Figure 1 summarizes the types and levels of knowledge.

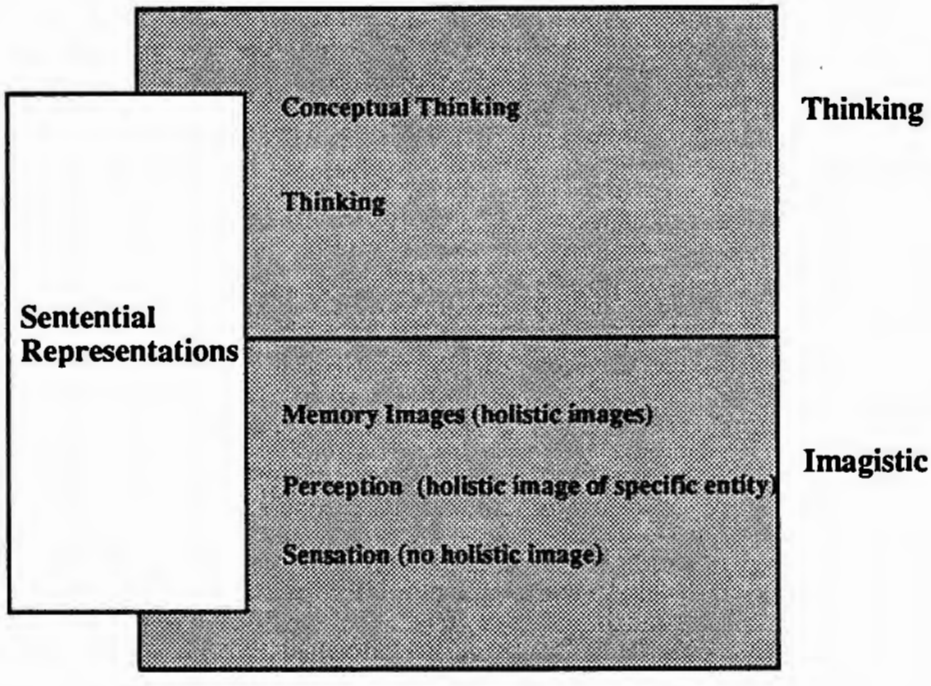


Figure 1: Levels of the Human Cognitive System

Perceptual and memory images reflect some specific object or experience or a hierarchically abstracted, idealized image of a class of objects or experiences such as a prototypical object.[4] In contrast, a sentential or verbal representation is not tied to any specific experience; it is an abstract representation. Thinking or reasoning processes require the manipulation of connections between the sentential representations and all related imagistic representations in a process of continuous reversible translation. Figure 2 illustrates some of the relationships between these elements of the cognitive system.

[4]Note that the term imagistic is used in contrast to the terms verbal or sentential, Images are stored in all modalities of experience.

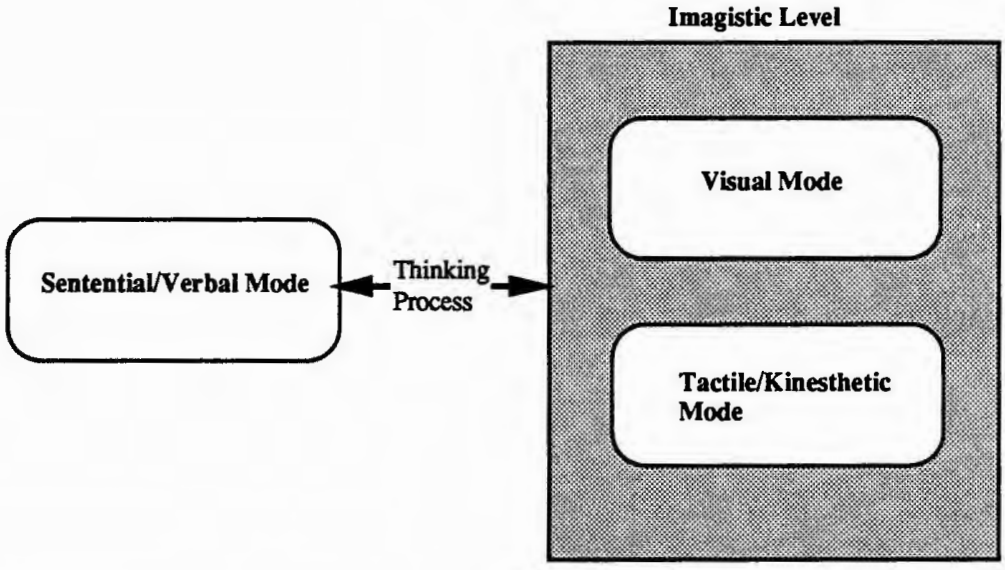


Figure 2: Interactions in the Cognitive System

Several characteristics of BDM-KAT are the direct result of this cognitive framework. In the current version of BDM-KAT the interface representations have been designed to take advantage of the verbal and nonverbal aspects of the knowledge structures being elicited. The initial set of tools implemented has been selected to draw upon knowledge structures common to all human information processing. In addition, an overall integration schema for the toolkit has been partially developed that includes additional tools and a "manager" component to assist the knowledge engineer in coordinating sessions and integrating the knowledge base. The set of tools that we expect to eventually comprise the complete toolkit will reflect the cognitive system as shown in Figure 3.

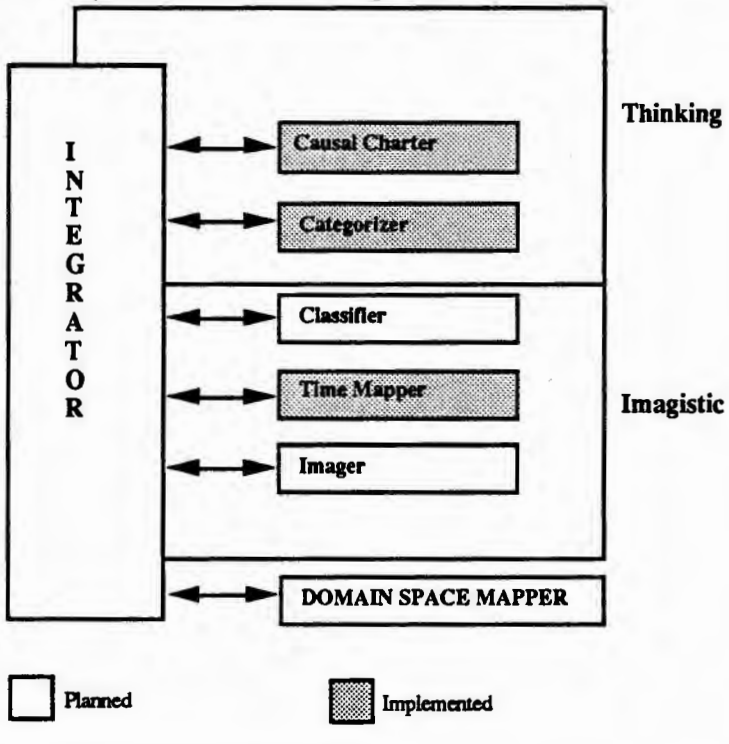


Figure 3: Conceptual Relationships in BDM-KAT

Experts tend to have developed stable knowledge representations that allow them to manipulate the knowledge to make inferences, to relate similar knowledge, and to be consistent in their statements, assumptions, and conclusions. However, this stability is often not seen and should not be assumed by the knowledge engineer. Vekker's theory suggest that questions, problems, and tasks be posed to the expert to stimulate the interaction of sentential and imagistic knowledge to demonstrate the stability in the translations between these two representations. Only then can the elicited knowledge be considered valid and acceptable across modes. In BDM-KAT the methods and procedures for using the tools have been developed to encourage the expert to draw upon his or her imagistic and sentential representations and, thus, to enable identification of the critical invariant relationships in the expert's knowledge.

Common Knowledge Structures

The ways in which human knowledge is structured and a knowledge of how these structures are organized has driven the selection and design of the initial set of tools for BDM-KAT. Expertise can rarely be elicited and represented in a single structure because humans tend to store knowledge about a domain in multiple structures. Tools to assist knowledge engineers in the process of eliciting knowledge should be capable of eliciting, translating, and representing multiple types of knowledge structures for a given domain. Complex, knowledge-intensive domains reflect diverse, yet interrelated types of knowledge. For the initial set of tools in the BDM-KAT prototype, we have focused on those recognized to be universally applied to verbal knowledge. Specifically, we have considered categorical, temporal, and causal structures.

Categorical Knowledge Structures

People tend to divide the world into relatively neat categorical structures. While for laymen the categorical divisions are frequently unstable (e.g., individual items may change their positions in terms of both category membership and typicality within a category (Barsalou, 1986)), experts in a specific field tend to hold stable and shared categorizations of their domain (Rosche, 1975). A substantial portion of one's understanding of a domain rests in knowledge about the items that appear in the domain, their individual and collective characteristics, and the relationships between them. Because this organization is natural to the domain expert, knowledge elicited within this structure is likely to be more easily obtained and more accurate.

Temporal Organization of Events

People also structure their knowledge into memories of events and orderly sequences of events. This knowledge appears to be organized internally into "mental time line models" in which the temporal relationships between episodes are specified (Barsalou, 1986). Where the episodes or process events can be decomposed to enabling or sub-events, hierarchical relationships exist between them similar to those between levels in a categorical taxonomy. This structural nature of event memory can be exploited in eliciting knowledge about processes that occur over time. Drawing from this model the domain expert can tap and share the order, temporal characteristics and relationships, and hierarchical associations among steps in the process.

Causal Structures in Knowledge

Finally, experts can be partially defined as those individuals who understand the why and how of a particular domain. All people, even novices, construct causal mental models of the world around them (Kelly, 1955). However, causal models held by a novice may reflect incorrect assumptions or focus (e.g., correlations), while those held by an expert tend to be more accurate. As with categorical and temporal organizations of knowledge, causal models can be elicited directly from an expert by focusing interactions on the expert's explanations for states and conditions in the domain of interest. Of these three types of knowledge, causal models are the most similar to the rules usually sought during the knowledge acquisition phase of knowledge-based system development.

BDM-KAT DESIGN AND IMPLEMENTATION

BDM-KAT is a toolkit that has been designed to enable a knowledge engineer to elicit knowledge based on both the structure and modality in which it is most likely to be represented in the domain expert's internal cognitive model. To accomplish this, BDM-KAT consists of three cooperating knowledge engineering tools: Categorizer, Time Mapper, and Causal Charter. Categorizer elicits knowledge about the stable, underlying structure of a scientific domain via the drawing and "filling in" of a tree-like, taxonomic structure. Time Mapper provides a context in which the domain expert positions items on time lines to express knowledge about the temporal characteristics of, and relationships between, events and subevents with a process. The knowledge engineer uses Causal Charter to elicit cause-effect knowledge for domain functions and events. Each of the tools provides the knowledge engineer and domain expert with a means for focusing attention on the aspect of the domain being elicited and graphically and textually recording and verifying information as it is elicited.

Each tool uses textual and graphic interface representations. All tools transfer information to, and interact with, a single database. The use of such a database also provides for a knowledge browsing facility while its contents are directly applicable to the development of knowledge documents. Figure 4 shows the high-level structure of the current tools. The following sections briefly describe the individual tools. The version described has been implemented on a Sun 3 system, successfully demonstrated, and has been used to elicit a significant knowledge base of hot isostatic pressing (HIP) information.

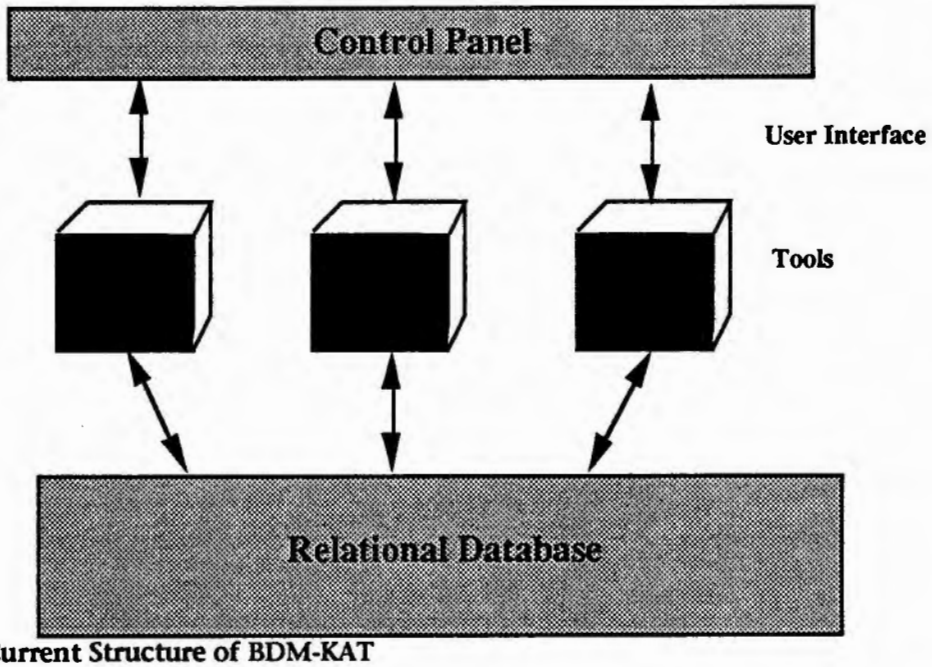


Figure 4: Current Structure of BDM-KAT

Categorizer

Categorizer elicits the underlying structure of a domain by requiring the expert to construct a taxonomy or set of taxonomies. As discussed earlier, categorical knowledge is commonly held for any domain or topic area. Furthermore, it is critical for constructing a domain model because the properties of specific objects and object classes will dictate much about how the knowledge can be used.

Using Categorizer the expert initially creates and names objects that exist in the taxonomy and places them at the appropriate position within the structure. The expert typically works in a top-down fashion, but the tool does not restrict the level at which objects can be named. After the taxonomy has been developed the essential, contingent, and second-order properties of each domain object are elicited from the expert. These properties will determine how the model can be perturbed during future verification activities. Object properties are best elicited in tandem with object taxonomies for two primary reasons. First, this sequence follows from the structure of the taxonomy itself. Second, while the expert's attention is focused on the taxonomy, she or he will be less likely to make errors in identifying properties for individual objects.

Categorizer creates and builds within the database a series of entities and relationships between them. The domain objects and properties specified during elicitation are represented individually in the database, as are their relationships.

Time Mapper

Whereas Categorizer elicits taxonomic and part-whole relationships among objects, Time Mapper elicits the structure of events or actions that occur during the execution of a process. Specifically, Time Mapper elicits the hierarchical relationships among events and subevents (at multiple levels), the overall duration of each event, and the specific starting and ending times of each subevent within an event. This elicitation effort is critical to developing a full process model because it provides both the decomposition of the process into well-specified components and the temporal relationships necessary for control. The expert is asked to name the process as a single event and then to name the subordinates of that event. This cycle iterates with increasing specificity, such that the subevents at one level are considered as events at the next level, until the events can be decomposed no further.

Causal Charter

The third tool, Causal Charter, is used to elicit high level "rules" and low level cause-effect chains. The rules are intrinsically implicit and are supplied by the expert through a description of an item to be "charted." The item under consideration may be either a function of a domain object (specified during Categorizer as a functional property of that object) or a subevent from the process decomposition state of Time Mapper. The structural elements in the item description are separated from other specified functions and are grouped together to indicate those elements that support each function in the description. Thus, the generation of clusters of structural elements (as units) are related to specific functions. The expert is asked to select those items from the description that provide support for each function given in the description. When specifying the cause-effect chains the domain expert selects the elements from the mechanism in the order in which they are caused or required by the mechanism. Thus, a completed chain indicates the process by which the final state or elements is reached.

CONCLUSIONS

BDM-KAT has been developed as a cognitively-based knowledge acquisition toolkit and as such, represents a major shift in the development of tools to support the knowledge acquisition process. Based on applications of cognitive research, BDM-KAT assists the knowledge engineer in solving the fundamental problem of eliciting and representing knowledge in a manner that reflects the expert's mental organization. Knowledge base content that is acquired in this manner is more likely to result in an expert system, intelligent tutoring system, or decision aiding system that can be used effectively in operational settings.

The overall concept of BDM-KAT calls for a more integrated set of tools and a coherent, integrated database representation of the elicited knowledge. Research in the context of BDM-KAT is continuing, with the emphasis on the development of capabilities that will realize the toolkit's potential for general purpose use. In pursuing future directions, the predictions and parameters of the human cognitive system, as modeled in the multi-dimensional model, will be used to define boundaries and requirements of the system.

REFERENCES

Anderson, J (1984) Representational types: A tricode proposal, Technical Report #82-1, Office of Naval Research, Washington, D.C.

Bainbridge, L. (1979) Verbal reports as evidence of the process operator's knowledge. International Journal of Man-Machine Studies, 11:4 (1979), 411-436.

Barfield, W. (1985). Expert-novice differences for software: Implications for problem-solving and knowledge acquisition. Behavior and Information Technology, 5(1).

Barsalou, L.W. (1986) The content and organization of autobiographical memories, Unpublished draft.

Dixon, N. (1981) Preconscious Processing. Chichester: Wiley.

Feigenbaum, E. (1977) The art of AI. Proceedings of the Fifth International Conference on Artificial Intelligence.

Gammack, J.G. (1987). Modelling expert knowledge using cognitively compatibls structures. In Proceedings of the Third International Expert Systems Conference, Oxford.

Hammill, B.W. (1984). Psychological issues in the design of expert systems. In Proceedings of the 2th Annual Meeting of the Human Factors Socety, San Antonio, TX.

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

Hoffman, R. (1987) The problem of extracting the knowledge of experts from the perspective of experimental psychology. AI Magazine, 8:2.

Kelly, G. The Psychology of Personal Construct, New York: Norton, 1955.

Kidd, A.L. (1983). Human Factors in expert systems. In K. Coombes (ed.), Proceedings of the Ergonimic Society Conference, Taylor and Francis, London.

McGraw, K. and K. Harbison-Briggs. Knowledge Acquisition: Principles and Guidelines, Englewood Cliffs, NJ: Prentice-Hall, 1989.

Michalski, R. Knowledge engineering through conceptual clustering: A theoretical framework and algorithm for partitioning data into conjunctive concepts. International Journal of Policy Analysis and Information Systems 4(3), 219-243.

Michie, D. (1982) The state of the art in machine learning. In D. Michie (Ed.) Introductory Readings in Expert Systems .

Nisbett, R. and T. Wilson. (1977) Telling more than we can know: Verrbal reports on mental processes. Psychological Review, 84, pp. 231-259.

Paivio, A. (1986) Mental Representations: A Dual Coding Approach, New York: Oxford University Press.

Rosche, E., C.B. Mervis, W.D. Gray, D. M. Johnson, and P. Boyes-Braem. (1975) Basic objects in natural categories. Cognitive Psychology, 8, 382-439.

Vekker, L.M. Mental Processes: The Agent, the Experience, Action, and Consciousness, Vol. 3, Leningrad University Press, 1981.

Vekker, L. M. Mental Processes: Thinking and the Intellect. Vol. 2, Leningrad University Press, 1976.

Vekker, L . M. Mental Processes. Vol. 1, Leningrad University Press, 1974.

Wielinga, B. and J. Breuker. Interpretation of verbal data for knowledge acquisition. In T. O'Shea, ed. Advances in Artificial Intelligence. New York: Elsevier, 1985,

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A PRACTICAL PROGRAM FOR HANDLING BIAS IN KNOWLEDGE ACQUISITION

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

ABSTRACT

While bias affects knowledge acquisition and validation, the field has developed little in the way of a program to counteract its influence. We propose a practical program for knowledge engineers to follow in their acquiring of expert knowledge. Steps are (1) anticipate which biases are likely to occur in the planned elicitation; (2) make both the knowledge engineer and the expert aware of the potential intrusion of these biases; (3) monitor the elicitation for their occurrence; (4) adjust, in real time, to correct for their occurrence; and (5) analyze the data to determine, more definitively, if they occurred.

1. INTRODUCTION

The presence of bias in expert judgment and in knowledge bases has recently been recognized, but few guidelines yet have been developed for handling the occurrence of bias in knowledge acquisition. Programs for handling bias are needed because bias affects both the gathering and validating of expert knowledge. Bias causes discrepancies that confuse and slow knowledge acquisition. Bias also enters into the knowledge base, degrades the quality of the knowledge, and, eventually, impairs the advice given by the knowledge-based system.

In this paper, expert knowledge will include only elicited knowledge, such as the experts' solutions to problems and their descriptions of their mental processes (definitions, assumptions, and algorithms) in arriving at solutions. The knowledge described in this paper will include that which the expert has given in numeric form, such as probability estimates, and that which the expert has been given in nonnumeric form, such as his descriptions of his assumptions. We view expert knowledge as dynamic, as changing as the expert receives new information (Ortiz, Wheeler, Meyer, and Keeney, 1989, p. 2). Gaines (1989) has provided a foundation for understanding how an expert's knowledge and performance evolve. He proposes that expertise is continuously developed through the expert's exposure to practical problems and to the data of others in his professional community. Through his consultations and access to other's theories and data sets, the expert receives practice and feedback, allowing him to refine his skills in eliciting data from clients, modeling the information, and giving advice.

The evolving nature of expertise has implications for knowledge acquisition. The knowledge acquisition process must reflect the changing nature of knowledge and allow for continual updating of the knowledge base or system (Gaines, 1989). Benbasat and Dhaliwal (1989) describe knowledge acquisition as occurring in phases where the knowledge base is repeatedly refined and where different kinds of validation are performed at the different phases. The program for handling bias being proposed in this paper is envisioned as fitting into this dynamic environment of evolving knowledge bases or knowledge-based systems. Particularly, the program can be used during any phase of knowledge acquisition when expert knowledge is being elicited. Then too, it

can be used as part of the validation process proposed by Benbasat and Dhaliwal (1989), particularly when the validation check calls for comparing the knowledge in the knowledge base or system to that of the expert source.

1.1. Two Views of Bias and Their Places Within Knowledge Acquisition

Artificial intelligence, decision analysis, and social science literature present two implicitly different views of bias. However, both of these views are relevant to knowledge acquisition. These two views differ in terms of the reference point from which bias is judged. The reference point for one definition of bias is the source expert's knowledge; the reference point for the other definition is *objective* measures of reality. In the descriptions below we have exaggerated the distinctions between the definitions for the purpose of illustration.

The first view, which will be termed *motivational bias,* proposes that *bias occurs when expressions of the expert's thinking do not match his actual thinking or his behavior at the time of the elicitation.* (In this paper, motivational bias will be limited to instances when the descriptions of the expert's thinking do not match his actual thinking; the expert's behavior, although an important reference point for defining motivational bias, is outside of the scope of this paper.) Thus, if the expert gives a different response than he currently believes because of comments from the knowledge engineer, this constitutes motivational bias. This view of bias comes from the soft sciences, particularly psychology and cultural anthropology. Proponents of this view consider the expert's way of thinking to be the *gold standard* that they wish to capture through the elicitation or the building of a knowledge-based system (Henrion and Cooley, 1987).

The second view, which will be termed *cognitive bias,* defines *bias as occurring when the expert's knowledge does not follow objective rules or standards considered to apply to the situation.* Thus, if the expert's predictions do not match what can be observed and measured, cognitive bias will have occurred, according to this definition. However, expert's knowledge often cannot be checked against known facts or physical measures of the phenomena. Thus, cognitive bias is frequently determined by checking the expert's data against the mathematical and statistical standards that apply to responses given in numeric form. To illustrate, if an expert gives probability estimates on all outcomes to a problem (previously defined as being mutually exclusive) and these probabilities do not sum to 1.0, these data will be considered biased because they do not follow normative statistical rules. This view of bias comes to knowledge acquisition from the fields of decision analysis and statistics (Mumpower, Phillips, Renn, and Uppuluri, 1987). The goal of this position is not to mimic the expert's thinking but to improve on it, such as by bringing the bias to the expert's attention for correction (Henrion and Cooley, 1987).

These views of bias are often implicit in knowledge acquisition applications or, at least, in particular phases of the application. We have observed that these views coincide with particular project goals, types of elicitation techniques, selections of expert response modes, and validation strategies.

The motivational view of bias, for instance, is more prevalent in knowledge acquisition projects whose goal is the psychological modeling of the source expert. In addition, proponents of this view of bias are likely to favor the use of *nondirective* interviewing techniques in the desire to avoid leading the expert. The verbal protocol technique, in which the expert thinks aloud but is not questioned by the knowledge engineer, is one such nondirective method. Also proponents of this view tend to avoid complex numeric response modes and allow the expert to respond spontaneously in his own, often nonnumeric, terms. Additionally, the motivational view of bias can be found in validations that focus on how well the knowledge base or the the knowledge-based system corresponds to the source expert's knowledge.

An emphasis on the cognitive view of bias is likely to coincide with projects whose goal is to improve on the expert's judgment. In projects where the concern with cognitive bias dominates knowledge acquisition, the expert is likely to be asked to respond using numeric forms, such as probabilities. Numerical responses can be more easily analyzed to determine if they are mathematically optimal. For instance, it is easier to mathematically determine that an expert is underestimating uncertainty if he is using probabilities than if he is simply stating that x is very likely. Thus, the formal checks on the expert's knowledge can be done from the very first phases of knowledge acquisition. The focus on validation from the cognitive bias perspective can also occur in later validation phases. For example, Gaines (1989, p. 52) describes one validation criteria of a knowledge-based system as checking *to what extent the knowledge in the system corresponds to actuality* (e.g., how well the knowledge corresponds to empirical data).

1.2. Causes of Bias

While the basic cause of both types of bias is the human being, the exact mechanisms by which it is induced differ. Motivational bias is induced by our human needs, such as for approval; cognitive bias by the way in which we process information.

1.2.1. Motivational Bias. In the determination of motivational bias, (1) the expert does not report what his solutions or thought processes actually were, (2) the knowledge engineer misinterprets the expert's report, or (3) the knowledge engineer misrepresents the expert's knowledge.

In the first aspect of motivational bias, altering of the expert's reporting, the phrasing of the questions could cause the expert to change his descriptions of his thinking. For example, if the knowledge engineer asks the expert if he used subgoal x in solving the problem, the expert may feel he should answer "yes," even if he did not solve the problem in this way. The mode in which the expert is asked to respond can also bias the expert's answer if he cannot accurately encode his final judgment into it. Some common numeric response modes are probability distributions, continuous linear scales, Saaty's paired comparisons, ratings, and rankings. Then too, the knowledge engineer's verbal or nonverbal responses can influence the expert's reporting. For instance, if the knowledge engineer leans forward, displaying intense interest in something that the expert is saying, the expert may unconsciously respond by exaggerating his statements on this topic. Other experts, in interaction with the expert, can have similar effects on the expert's thinking. Furthermore, experts' thinking can be affected by their imagination of how those not physically present, such as clients or supervisors, might view their responses. For example, in a study examining whether magnetic fusion projects would meet their technical milestones, some of the experts said that they considered their organization's judgments in giving their estimates (Meyer, Booker, Cullingford, and Peaslee, 1982).

There are several reasons why experts' reporting of their thoughts can be influenced by others. First, most people have an emotional need to be accepted and to receive approval (Zimbargo, 1983). Secondly, people are generally unaware of how they make decisions, such as in solving problems (Hogarth, 1980, p. ix). Yet, it is difficult for them to admit ignorance because Western scientific tradition assumes that the problem-solving process can be precisely stated (Denning, 1986, p. 345). Thus, in elicitation situations, the expert is likely to be responsive to what he believes the knowledge engineer, his managers, or other experts wish to hear. Their expectations or wishes are communicated, often unconsciously, through their questions, responses to the expert, and body language. The expert is likely to acquiesce to suggestions of an acceptable answer or to means by which he might have solved the problem.

The expert's verbalizations can be biased by another source--the knowledge engineer's interpretation of his words. We, as humans tend to perceive and interpret incoming information in a selective way that supports what we already believe. Sometimes, this leads to a

misinterpretation, or biasing, of the information. For instance, when the expert mentions a new term, we tacitly assume a meaning based on our experience with similar sounding words. Likewise, we may think that we hear the solution that we expect. Knowledge engineers can be particularly prone to this bias because of a method that they have used for learning the expert's field. This method was to study the expert's domain, to build mental constructs representing the knowledge, and to understand the expert's knowledge structure by comparing to the knowledge engineer's own. This comparative means of learning the expert's knowledge structure would seem prone to bias because the knowledge engineers interpret the expert's thoughts through the filter of their own.

The elicited knowledge can also be altered by the representation of it. Expert data are often modeled for analysis, such as for detecting bias or for knowledge-based systems. The individual performing these tasks makes many tacit decisions and assumptions about the data's appearance and performance. For example, when a model is selected for the data, the model assumes particular properties of the data, such as its distribution. These assumptions may not be valid. Then too, the knowledge engineer may have to aggregate multiple and differing expert judgments to provide one input value. Mathematical aggregation schemes often require assumptions, and different ones, like the mean versus the weighted average, can produce different answers. The knowledge engineer also makes decisions about the organization of the knowledge in the system and its representation that determine how the data are implemented.

1.2.2. Cognitive Bias. In the definition of cognitive bias, bias is a consequence of the way in which we think. The following are some characteristics of the way that we think.

Humans model their world to understand, predict, and control it (Clancy, 1989, p. 11). It is important to note that there cannot be a perfect match between the model that the expert uses in problem solving and the reality being modeled. We selectively take in information, often perceiving those data that support rather than contradict our beliefs.

Probably the human mind is limited in how much information that it can process and in how much it can remember Hogarth (1980, p. 9). In order to reduce the cognitive burden, people tend to take short cuts when solving a complex problem. Thus they start with a first impression and integrate the information in a sequential manner, making only a few minor adjustments. Later, if additional information is received, they probably will not adjust their initial impression to give a more accurate judgment. In other words, if an individual who has already reached an initial solution is given contradictory data, he will probably not take this data sufficiently into account when generating a final answer. In particular, this sequential means of integrating information handicaps us in making predictions where large or sudden changes are involved. This limiting effect is called *anchoring* or *anchoring bias*.

The human mind has limited memory capacity for information processing. As Miller (1956) has noted, most individuals can not discriminate between more than seven things at a time. This limitation in information processing causes people to be inconsistent in working through the problem. For instance, people commonly forget an assumption made earlier and contradict it, thus causing *inconsistency bias*.

Then too some data is more easily recalled than others. For instance, data involving catastrophic, familiar, concrete, or recent events may be easier to recall (Cleaves, 1986; Gold, 1987). This effect, termed *availability bias*, can lead to the overestimation of the frequency of some events.

1.3. Impact of Bias on Knowledge Acquisition

Bias is of concern to knowledge acquisition because it slows the process. To illustrate how bias can complicate acquisition, particularly elicitation, an example of motivational bias is used.

Suppose that the knowledge engineer asks the expert if he mentally models the problem using a decision tree structure. The expert may answer "yes" even if he did not use this means of modeling the problem. The knowledge engineer then faces the difficulty of resolving the discrepancies that are likely to arise between the expert's claim and his solutions (Meyer, Mniszewski, and Peaslee, 1989). These discrepancies can confuse and slow the acquisition process.

Bias affects validation of knowledge bases and knowledge-based systems whether bias is judged from the standard of the expert's problem-solving processes or from the standard of statistics and logic. Motivational bias also interferes with the eliciting and modeling of expert knowledge for the knowledge bases or systems. For example, if the knowledge engineer misinterprets and then misrepresents the expert's knowledge, the elicited knowledge will not match that of the source at the time of the elicitation. Cognitive bias hampers obtaining knowledge that follows particular standards or matches empirical data. For example, a plant pathologist gave a higher expected value to fungicide treatment than the sensitivity analysis performed on the knowledge base yielded (Henrion and Cooley, 1987, p. 475). The analysis showed that the fungicide treatment had a lower value than the expert judged because the infection is difficult to cure and the fungicide not highly effective. If cognitive bias is not checked, the implemented knowledge-base system can provide advice which is not mathematically optimal.

In general, bias needs to be treated because of its negative impact on other's perception of expert knowledge and knowledge-based systems. Several articles have recently questioned the accuracy of expert clinical judgments in psychology and psychiatry (Faust and Ziskin, 1988; Dawes, Faust, and Meehl, 1989). The credibility of expert knowledge is open to this same sort of questioning if no attempts have been made to examine and counter bias.

Section 2 of this paper describes how to handle bias using five steps. Guidance is given on how to follow each step using examples from both motivational and cognitive bias. A summary of the work is given in Section 3.

2. PROGRAM FOR HANDLING BIAS

Approaches to handling biases are rare (Cleaves, 1986) and in their early stages. They are perhaps as much art as they are science. The program proposed here is no exception. One reason that there are not more programs for handling bias is that it is a difficult topic to study. Studying, much less trying to counter bias, is complicated by not having a readily available baseline from which to determine the direction and magnitude of the bias. For the questions that expert knowledge is elicited, there are frequently no known single *right* answers or empirical data. Thus, the expert's data cannot be simply compared to an answer looked up in a reference or to the result of modeling data. Measurement of motivational bias is further complicated by the absence of objective standards for comparison. At least with cognitive bias there are traits of the expert's answers that can be compared to standards. For example, do the expert's exclusive but dependent probabilities sum to zero as they should? With motivational bias, the baseline of the expert's knowledge, has its description not been altered, is difficult to determine, especially because expertise is not static. As Rosenfield (1988, p. 194) argues in his book on memory, higher mental functions are not fixed procedures but subject to constant reconstruction. While we recognize that both biases are difficult to detect, we believe that for progress to occur programs like this one must be proposed and applied to knowledge acquisition.

Our approach differs from the one presented by Cleaves (1986). First, Cleaves' proposal focuses on cognitive bias; second, Cleaves tries to anticipate biases by the judgment processes in which they are likely to occur, namely hypothesis and solution generation, decision rule articulation, uncertainty assessment, and hypothesis evaluation. While we agree that biases occur during these

processes, we try to anticipate the biases by the elicitation components that are likely to exhibit them. We assumed that many knowledge engineers were lay persons in the areas of human cognition and that this approach might be easier for them, at least as a starting point.

Another major difference in our program is its real-time emphasis. Given the evolving nature of expertise, bias is best detected when it is being elicited. This program stresses monitoring and adjusting for bias, particularly motivational bias, in real time rather than mathematically compensating for it afterwards. It is much more difficult to determine that motivational bias has occurred after the elicitation because the baseline--the expert's knowledge--is likely to have changed. For this reason, we consider each elicited datum to be a snapshot that can be compared to a snapshot of the expert's state of knowledge at the time of the elicitation.

Our proposed program consists of these general steps:

Step 1. Anticipate the biases to which the planned elicitation is prone.
Selected biases are listed according to the situations in which they are likely to occur. If the exercise of anticipating the biases leads the reader to wish to redesign the elicitation at this point, step 4 offers some ideas. The reader could look under step 4 for means of countering the likely biases and integrate them into the design of the elicitation.

Step 2. Make the knowledge engineer and the expert aware of the potential for particular biases.
The knowledge engineer needs to be aware of her own potential for influencing the expert, misinterpreting the expert's words, or misrepresenting the knowledge. The expert needs to be informed about the biases he is likely to exhibit given the elicitation situation. For example, the expert could be told about the tendency to underestimate uncertainty in making predictions. The expert then has a chance of giving answers that more fully encompass the actual amount of uncertainty.

Step 3. Monitor the elicitation for the occurrence of bias.
For each of the selected biases, signs that they may be occurring are given. For instance, if experts in a group situation are deferring to another individual, this could signal that their judgments are being influenced by that individual.

Step 4. Adjust, in real time, to counter the occurrence of bias.
Some techniques are given for modifying the elicitation process to prevent particular biases from intruding. Often this involves controlling the factors which contribute to the bias, such as fatigue leading to increased inconsistencies. Other times, it requires the encouragement of an opposite bias to counter the occurrence of the first bias. For example, the tendency to be unconsciously influenced by others in the group can be countered by encouraging the expert to anchor to his first impression.

Step 5. Analyze the data for the occurrence of particular biases.
Examples are given of biases that we have tested for. If step 5 is the only step of the program being followed, the analysis will necessarily be simpler than if steps 3 and 4 were also followed. If these other steps were followed, they would provide the additional data needed for performing more complex analyses. In general, adequately testing for one of the motivational biases requires this more complex testing. Occurrence of a cognitive bias, such as the underestimation of uncertainty, can often be determined by simple mathematical tests.

While the view of bias, motivational or cognitive, is often implicit in a knowledge acquisition project, we recommend that the reader consciously select which view or views of bias apply. There are several reasons for consciously selecting the view or views of bias. First, the choice can

be made with the consideration of which view will best serve the application, the phase of knowledge acquisition, or validation phase. For example, if the focus of the project is learning the expert's problem solving in order to emulate it, the motivational definition of bias would be appropriate. If the project involves estimating the likelihood of future events, the cognitive definition would be a natural choice. People are inaccurate in making predictions, and the cognitive view of bias would help counter this weakness. Second, the reader needs to recognize that use of one definition could cause bias according to the other definition. For example, use of the cognitive definition would propose that a mathematically incorrect judgment be modified. This act would cause a misrepresentation of the expert's data, a bias, according to the motivational definition. Thus, we recommend that the reader formulate rules for when each view of bias will be applied, if more than one view is to be used. To illustrate, the definition of motivation bias could be used in acquiring knowledge for building the conceptual knowledge base and the cognitive definition used thereafter.

This program for countering bias is to be used during the elicitation of knowledge for conceptual or elicited data bases or for implemented knowledge-based systems. The program can be applied after the basic method of elicitation has been selected. At this point, the knowledge engineers and project personnel are ready to think about the biases to which their elicitation method is prone.

There are several options for those wishing to follow this program. Different steps can be followed depending on the knowledge engineer's time and interest. For example, if the knowledge engineer wishes to do the minimum in treating bias, she can simply follow step 5--analyze the data for the occurrence of a particular bias and test for its presence. If, on the other hand, the knowledge engineer would like to follow more but not all of the steps, she could do steps 1 and 5, or 1, 2, and 5, or 1, 3, and 5, or 1, 2, 3, and 5. We suggest that step 5, analyzing for bias, always be done regardless of the other steps. Step 5 could be considered one of the checks done as part of the ongoing validation of knowledge acquisition.

2.1. Step 1--Anticipating Potential Sources of Bias in the Elicitation

Anticipation of potential sources of bias is the first step in the proposed program. This process is illustrated using six frequently encountered biases. These were selected because they represent a range of bias within the two definitions. Certainly these are not the only sources of bias. However, they are ones that have frequently been encountered and for this reason are used as a starting point.

The elicitation methods mentioned below have been broken into their components to allow a clearer listing of biases. The knowledge engineer can determine the parts of her planned elicitation and then proceed to look up the biases to which they are prone in the index below. The knowledge engineer can turn to the definitions of bias in step 1 to read further on these sources. These definitions provide information on why particular biases are likely to occur. If the knowledge engineer wishes to carry this step, anticipation of bias, further, she can use Step 1 information to modify the planned elicitation. Step 4 also contains some information that could be used to this end.

2.1.1. Components of Elicitation.

1. The elicitation situation is where the knowledge engineer communicates with the expert. There are three distinct situations which can overlap each other in practice.

The individual interview: the knowledge engineer meets with an expert in a face-to-face situation. The individual interview could be focused, structured, or unstructured in its questioning and include problem discussion, description, or analysis, as described by Boose and Gaines (1988, p. 38-39). Most of the interviewing techniques, even those

for eliciting detailed problem-solving data (e.g., protocol analysis, the verbal probe or the ethnographic method), can be used in this situation.

The interactive group: the knowledge engineer meets with the expert in the presence of other experts. Usually the knowledge engineer moderates these sessions and leads the questioning. This elicitation situation can range from traditional meetings, to negotiations, to decision-analysis sessions, and to interactions of users on terminals in a computer network (McGraw and Harbison-Briggs, 1989, p. 263). The interactions of the group members can be structured to differing degrees. An example of an unstructured interactive group would be the traditional business meeting; a highly structured interactive situation would be one in which the experts were requested to provide their responses to each other in specific forms at particular times. For instance, experts in army export control were required to discuss the problem and then give their data in the form of estimates with a few sentences explaining their reasoning (Meyer and Johnson, 1985). Elicitation techniques that provide broad coverage (e.g., the verbal probe), as opposed to in-depth probing, are best used in the interactive group situation.

The Delphi technique: the knowledge engineer keeps the experts physically separate but acts as a channel for distributing each expert's information to all of the experts. (This technique was developed by Rand Corporation to avoid having members of the group influence each other's responses.) In strict Delphi, the knowledge engineer obtains the experts' responses through the mail, makes these responses anonymous, redistributes these anonymous responses to the individual experts, and allows the individual experts to revise their previous responses. This process can be repeated as many times as desired. In general, the Delphi has not been used to elicit in-depth problem-solving data because this type of data is best elicited when the knowledge engineer and expert are face to face. Instead, structured survey types of questions are asked in the Delphi situation. The Delphi has been used for classifying problems and for group negotiations (Boose and Gaines, 1988, p. 40).

2. The modes of communication are the means--face to face, telephone, or mail (electronic or postal)--by which the knowledge engineer and expert communicate during the above-mentioned elicitation situations. Several modes can be combined in one elicitation process. For example, the knowledge engineer can send the expert the problem, answer the expert's questions about it over the telephone, and then elicit the expert's problem solving in person. In addition, modes can sometimes be substituted for one another. For example, in the Delphi situation, the information could, if it were simple and brief, be communicated by telephone rather than by mail.

3. The response mode is the form in which the expert is asked to give his knowledge. Frequently in knowledge acquisition the expert is allowed to respond naturally rather than encode his thoughts into a form selected by the knowledge engineer. Other times, a numeric response mode, such as probabilities, continuous linear scales, Saaty's paired comparisons, ratings, uncertainty measures, and confidence ranges, is requested.

4. The aggregation scheme is the means by which multiple and differing expert's responses are made to form one response, such as one line of reasoning or a numerical solution. The two major techniques are mathematical aggregation and behavioral aggregation. **Mathematical aggregation** is the use of mathematical means to combine multiple expert's quantitative answers into one answer. Some mathematical methods weight the expert's answers equally (e.g. the mean), others weight the expert's data differently in attempts to give higher weights to the *more expert* data. Mathematical aggregation will not be addressed below because it is outside the scope of this paper. **Behavioral aggregation** relies on the experts reaching a consensus (Seaver, 1976). The aggregation occurs during, rather than after, the elicitation session. When behavioral aggregation is used in interactive group situations, the experts are informed that they must reach a consensus and often employ persuasion and compromise to do so. When behavioral aggregation is used with the Delphi technique, the experts give their responses and receive feedback until they reach one response.

2.1.2. Index of Selected Biases.

Components	View of Bias--Source
Elicitation Situations	
Individual Interview	Motivational--Social pressure from KE Cognitive--Inconsistency
Delphi	Cognitive-Inconsistency Cognitive--Anchoring
Interactive Group	Motivational--Social pressure, group think Cognitive--Inconsistency
Modes of Communication	
Face-to-Face	Motivational--Social pressure from KE Cognitive--Underestimation of uncertainty
Telephone	Motivational--Social pressure from KE Cognitive--Availability Cognitive--Anchoring Cognitive--Underestimation of uncertainty
Mail	Motivational--Social pressure, impression management Motivational--Misinterpretation by KE Cognitive-Inconsistency Cognitive--Availability Cognitive--Anchoring Cognitive--Underestimation of uncertainty
Response Modes	
Complex, numeric response modes, such as probability, distributions, especially if the expert is inexperienced in their use.	Motivational--Misinterpretation by expert Cognitive--Inconsistency Cognitive--Underestimation of uncertainty
Aggregation	
Behavioral Aggregation	Motivational--Social pressure, group think

2.1.3. Definitions of Selected Biases.

2.1.3.1. Motivational Bias--Altering of the expert's responses through social pressure (from the knowledge engineer, group think, or impression management) or interpretation.

Social pressure. Social pressure is the altering of the expert's descriptions of his thoughts arising from the desire to be accepted and to see himself in the most positive light possible. This altering can take place consciously or unconsciously. The social pressure can come from those physically present, such as the knowledge engineer or the other experts, or from the expert's own internal evaluation of others' reactions.

Social pressure from the knowledge engineer is most likely to occur in those methods where she is meeting in a face-to-face situation with the experts, such as in the individual interview and the interactive group. In face-to-face situations, the knowledge engineer can intentionally or unintentionally influence the expert through her body language, facial expression, intonations, and

choice of words. We expect this source of bias to be weaker in telephone conversations and weaker still in communications by mail. These last two communication modes do not allow some of the above-mentioned means of expression that the face-to-face mode does. In addition, social pressure bias is more pronounced when the knowledge engineer is asking leading questions. Thus, it is weaker when the knowledge engineer is using one of these methods--the verbal protocol, verbal probe, or ethnographic technique (Meyer, Mniszewski, and Peaslee, 1989). The verbal protocol requires the expert to think aloud and thus avoids leading the expert because it does not involve questioning him; the verbal probe uses general, nonleading phrases; and the ethnographic techniques uses the expert's own words in formulating questions.

Social pressure from others in the group induces individuals to slant their responses or to silently acquiesce to what they believe will be acceptable to their group (Meyer 1986, p. 89). Zimbargo, a psychologist, explains that it is due to the basic needs of people to be loved, respected, and recognized that they can be induced or choose to behave in a manner which will bring them affirmation (1983). There is abundant sociological evidence of conformity within groups (Weissenberg, 1971). Generally, individuals in groups conform to a greater degree if they have a strong desire to remain a member, if they are satisfied with the group, if the group is cohesive, and if they are not a natural leader in the group. Furthermore, the individuals are generally unaware that they have modified their judgments to be in agreement with the group.[1]

Group think. One mechanism for this unconscious modification of opinion is explained by the theory of cognitive dissonance. Cognitive dissonance occurs when an individual finds a discrepancy between thoughts that he holds or between his beliefs and actions (Festinger, 1957). For example, if an individual holds an opinion that conflicts with that of the other group members and he has a high opinion of the other's intelligence, cognitive dissonance will result. Often the individual's means of resolving the discrepancy is by unconsciously changing his judgment and/or the reporting of that judgment to be in agreement with that of the group (Baron and Byrne, 1981). For example, Janis' study (1972) of fiascoes in American foreign policy illustrates how presidential advisors often silently acquiesce rather than critically examine what they believe to be the group judgment. This phenomena has also been called group think bias, the follow-the-leader or bandwagon effect.

Group think is only likely to be a concern in an interactive group situation. It is further likely to occur in situations where behavioral aggregation is used because this type of aggregation requires that pressures toward conformity be encouraged.

Impression management. Another type of social pressure can occur as the expert imagines the reactions of those not physically present. This effect can occur in any elicitation situation. However, it seems to be more noticeable when it is not covered by other effects, such as social pressure caused by the knowledge engineer. For this reason, its occurrence is most noted in mail surveys or Delphi situations. The individual may try to answer in such a way as to bring the most approbation, such as from the person who has written the questions. Then, too, he may try to respond in such a way as would be acceptable to his employer or to society in the abstract. For this reason, this source of social pressure has been termed impression management (Goffman, 1959). Payne (1951) has found evidence of individuals giving the responses that they perceived to be the most socially acceptable rather than those which accurately portrayed their thoughts or actions. For example, on surveys, people claim that their educations, salaries, and job titles are better than they are. Often there is a 10% difference between what is claimed for reasons of prestige and what objectively *is* (Meyer 1986, p. 90).

---

[1] Note that we are making a distinction between unconscious conformity and an expert changing his mind as a result of new information received. The former is group think, the later an illustration of knowledge in action.

Misinterpretation. Misinterpretation is the altering of the expert's responses as a result of the methods of elicitation and documentation. While this effect is prevalent, it has not received much attention. Frequently misinterpretation occurs as a result of the response mode. If the expert can not adequately translate his judgment into the response mode, misinterpretation will result. We have noticed that experts seem to have more difficulty with the following response modes: probability distributions, ranks, and percentiles.

Misinterpretation is also more likely with elicitation and documentation methods that are written from the knowledge engineer's, rather than the expert's, viewpoint. For example, we have all had the frustrating experience of trying to force fit our views into the limited response options of a mail survey.

2.1.3.2. Cognitive Bias--Data failing to follow mathematical and logical standards because of inconsistency, anchoring, availability, or underestimation of uncertainty.

Inconsistency. Inconsistency is the inability to be consistent in one's solving of problems, especially through time. Of all of the biases mentioned here, this is the most common. Individuals often unintentionally change definitions, assumptions, or algorithms that they meant to hold constant throughout the problems. Inconsistency in an individual's judgment can stem from his remembering or forgetting information during the elicitation session. For example, the individual may remember some of the less spectacular pieces of information and consider these in making judgments later in the session, or the individual may forget that particular ratings were only to be given in extreme cases and begin to assign them more freely toward the end of the session.

As Dawes, Faust and Meehl (1989, p. 1671) have noted, such factors as fatigue, recent experience, or seemingly minor changes in the ordering of the information or in the conceptualization of the "...task can produce random fluctuations in judgment. Random fluctuation decreases judgmental reliability and hence accuracy." These inconsistencies may result in answers that do not make logical or Bayesian sense. For instance, a series of answers proposing that factor A was more critical than B, B more than C, and C more than A would not make logical sense. Similarly, if an expert gave the same probability of A for two situations: one of which involved an influential factor C and one which did not, his answers would not be coherent from a Bayesian viewpoint.

The natural tendency toward inconsistency is acerbated by several conditions such as memory problems, confusion, and fatigue. During elicitation sessions of more than 30 minutes, people often forget the instructions, definitions, or assumptions that they were requested to follow. For example, the experts may forget that a rating of nine meant a near certainty and assign it more easily than the definition specified. Thus, unstructured elicitations, which do not have periodic reviews of the problem information, are more likely to have high inconsistency. This inconsistency can be between experts' answers (e.g., the experts meant different things by the same numerical answer) or within an expert's answer (e.g., sometimes the expert gave a specific rating more easily than at other times). Also, situations where the expert's understanding through time cannot easily be monitored are more prone to inconsistency. These situations include the Delphi or mail survey.

Confusion can also lead to inconsistency. Thus, any of the more complicated response modes, such as probability distributions and percentiles, are more prone to this problem. This confusion is why training the expert in the use of these modes is recommended. In addition, if the experts must mentally juggle more than five to seven things, such as in rating them, they are likely to become confused and inconsistent. It is for this reason that the Saaty paired-comparison mode is used even though it is more time consuming than some of the other response modes.

Anchoring. Anchoring is the failure to adjust sufficiently from one's first impression in solving a problem. We would rate it next to inconsistency in terms of frequency of occurrence. Sometimes this tendency is explained in terms of Bayes Theorem as the failure to adjust a judgment in light of new information as much as it would be in terms of Bayes mathematical formula (Meyer, 1986, p. 88). Spetzler and Stael von Holstein (1972) and Armstrong (1981) describe how people tend to anchor to their initial response, using it as the basis for later responses. Ascher (1978) has found this problem to exist in forecasting where panel members tend to anchor to past or present trends in their projection of future trends. Ascher determined that one of the major sources of inaccuracy in forecasting future possibilities, such as markets for utilities, was the extrapolation from old patterns that no longer represented the emerging or future patterns. Another example of anchoring occurs when a member of a group's last estimate is closer to his initial impression than it would be had he fully taken earlier group discussions into account .

Anchoring is most prevalent in situations where the expert is not likely to experience the opposite bias of being influenced by the knowledge engineer or the group, such as in the Delphi method. In addition, those modes, such as mail or telephone communications, where the expert's thoughts cannot be easily monitored by having the expert think aloud, are prone to this bias. Also, we have noticed that experts are more likely to stick with their anchor if they have either described it orally or in writing and fear losing face for changing their mind.

Availability. Availability bias arises from the differing ease with which events can be retrieved from long-term memory. Data involving catastrophic, familiar, concrete, or recent events tend to be easier to recall. Availability bias affects people's ability to accurately estimate frequencies and recall other aspects of the event. For example, the incidence of severe accidents in reactors tends to be overestimated in part because of their catastrophic and newsworthy nature.

Availability bias is more common when the expert does not receive any information from others and, thus, does not have a chance of triggering other, less accessible, memory associations. For this reason, the individual interview is the most prone to availability bias, and the interactive group, the least. With individual interviews, a series of different scenarios is often used to help the expert enlarge on the sample of things contributing to his final answer.

Availability bias is also more common with telephone and mail modes of communication because the expert is usually not given much background before being asked point blank for the answer. A structured hierarchical presentation of the information, such as from the general to the specific, can alleviate this weakness.

Underestimation of uncertainty. People will underestimate the amount of uncertainty in the answers that they give. For example, when people are asked to put a range around an answer such that they are 90% sure that the range encompasses the correct answer, their ranges only cover 30-60% of the total (Capen, 1975). Even when they are given quizzes and feedback on their performance, they cannot break the barrier of covering only 70% (Capen 1975, p. 846). A popular explanation for this effect is that we are uncomfortable with the amount of uncertainty in life, and thus, minimize it. In particular, we may avoid confronting the large uncertainties in our judgments.

Although this effect is very widespread, Martz, Bryson, and Waller (1984, p. 72) have noted that it is more pronounced with probability and chance estimates than with some of the other response modes. Chance estimates, also called odds, are given as 1 chance in a total, such as 1 in 1000.

2.2 Step 2--Making the Knowledge Engineer and the Expert Aware of the Potential for Bias

As a result of step 1, the knowledge engineer has become aware of her elicitation's tendencies toward bias. The experts also need to be made aware of the biases to which they may be vulnerable. This can be conveniently done when they are being briefed about the elicitation.

The experts need to be (1) informed about the biases, and (2) acquainted with the elicitation procedures. Regarding the biases, anticipated biases can be defined and their contributing factors discussed with the experts (see definitions in section 2.1.3). It should be noted that making the experts aware of the biases helps but does not completely alleviate the problem. In some cases, the cause of the bias, such as with the underestimation of uncertainty, is too ingrained to be completely overcome. In other cases, the experts do not make the needed effort to counter the natural tendency toward bias. People typically believe that others, not themselves, will suffer from the biases described. With some biases, such as anchoring and underestimation of uncertainty, the experts can participate in test situations designed to evoke the bias. The knowledge engineer can read the answers and allow the experts to score their own. This kind of demonstration can convince the expert that he too is prone to the bias.

The experts also need to know about the elicitation procedures. If they are confused about how and when they are to respond, the data gathered as well as the expert's cooperativeness can be negatively affected. The expert may be confused by the unfamiliar response modes; these should be rehearsed in a training session. [If the response mode is probability distributions, Hogarth (1980, p. 149) provides eight keys to its use.]

2.3. Step 3--Monitoring the Elicitation for the Occurrence of Bias

For many of the selected biases, there are signs that indicate their occurrence. The list below is a partial one based on our experience. The knowledge engineer or a trained observer can watch for these signs during the elicitation. In general, monitoring biases, as described here, requires that the experts verbalize their thoughts and answers. Without this feedback, we have found the monitoring to be much more difficult.

Listed below are signs of selected biases.

Group think. There are several signs that a group-think situation may be developing. Generally, no difference of opinion is voiced, and the experts appear to defer to another member of the group or to each other (Meyer, 1986, p. 95).

Inconsistency. A number of signs can indicate inconsistency. The knowledge engineer can hear many of these, if the experts are verbalizing their thoughts and answers. In particular, she can detect when a response mode or rating is being applied more easily through time (Meyer, 1986, p. 94). Experts tend to apply the extremes of a rating scale more easily as they become fatigued. The knowledge engineer can also hear when the expert is contradicting an assumption that he made earlier. For example, a tank expert chose two very different routes through the mapped terrain because the second time he unconsciously assumed that his company was the main effort and *had to push hard.*

Inconsistency can also be monitored by the use of Bayesian-based scoring and ranking techniques. During the elicitation, the expert's judgments can be entered into a scoring and ranking program, such as that of Saaty's Analytical Hierarchical Process (1980), to obtain a rating of their consistency. Then, if this number is too high, indicating significant inconsistency, the experts can redo their judgments as described in step 4.

Availability. A potential problem with availability bias is indicated if the expert does not mention more than one or two considerations in giving his answer. If the expert only considers a few things, these were probably the most easily remembered and the answer is likely to be skewed to reflect these few.

Anchoring. Anchoring bias can be suspected if the experts receive additional information from experts or other sources during the elicitation but never waiver from their first impression. For example, reactor code experts were asked to compare the performance of their computer codes to plots of experimentally generated data. Often they commented on their first impression. When they examined the plots more closely, they typically found places where the computer code did not capture the experimental phenomena. However, the experts usually simply adjusted upward or downward of their initial assessment rather than revising it completely (Meyer and Booker, 1987).

2.4. Step 4--Adjusting, in Real Time, to Counter the Occurrence of Bias

This step is perhaps the most delicate one because if done carelessly it could confuse the results and any later analyses. The knowledge engineer needs to decide in advance on the timing of the adjustment because these adjustments change the conditions under which the data is gathered. If a condition leading to bias is corrected before the analyzable data has been gathered, there is no problem. If, however, the analyzable data has been gathered under two conditions, when bias was occurring and then when it was corrected, the data will be mixed. Unless the situations can be clearly separated, such as before and after correction of bias x, the knowledge engineer will not be able to later test for the presence of this bias.

Mentioned below are some of the techniques that we have used to adjust for bias. In general, our approaches have been to (1) impede those factors contributing to a particular bias, or (2) to employ the opposite bias. For example, the knowledge engineer's tendency to lead, or influence, the expert contributes to social-pressure bias. When using the first approach, we have advocated the use of elicitation techniques--verbal protocol, verbal probe, ethnographic technique--that curb this tendency (Meyer, Mniszewski, and Peaslee, 1989). Another example of this approach is focusing on the fatigue and confusion that contribute to our natural tendency to be inconsistent. The elicitation sessions can be scheduled to stop before the point when most of the experts become fatigued. The basis of the second approach, fighting bias with bias, comes from Stanley Payne (1951), the grandfather of survey design. Payne believed that all interviewing was biased and that one should therefore aim for equal but opposite biases. An example of this technique is to try to have experts anchor to their own judgments in attempts to counter a group-think situation.

The following are suggestions on how to adjust for selected biases.

Group think. Social pressure from group think can be countered using techniques from both the approaches (1) and (2) above. (Also see Meyer, 1986, p. 95-96). Using the first approach, the knowledge engineer can stop the elicitation and warn the group members about group think. If there is an official or even a natural ex officio leader in the group, that individual can be asked to give his responses last, or privately, so as not to influence the other group members. In addition, if someone other than the knowledge engineer has been leading the group meeting, he can be encouraged to be nondirective during the meetings. An explanation of the group-think phenomena usually convinces them that better discussions and data will result from their avoiding *leading*.

The other approach is to try to counter the effects of group think with anchoring. One technique for fostering anchoring is to require the group members to write down their judgments and reasoning. In this way, they are more likely to anchor to their own judgments rather than silently acquiesce to someone else. If the experts are to discuss their judgments, each person can record and report his before the floor is opened to discussion. Once individuals have publicly announced

their view, they are unlikely to spontaneously modify it . (They will still modify their view if someone raises a valid point that they had not previously considered.)

Inconsistency. Inconsistency can be reduced by using several techniques. The first is to address the aspects of the elicitation that are contributing to the inconsistency. As mentioned earlier, fatigue is a contributor to inconsistency. If the knowledge engineer has noted that the experts are becoming more inconsistent with time, she can quickly end the meeting or schedule a break. In general, two hours is the maximum amount of time that experts participate in discussion or problem solving before becoming tired. (Experts often signal their fatigue either by briefer responses or by leaning way forward or back in their chairs.)

Another contributor to inconsistency is faulty memory. If at the beginning of every session the statement of the problem, definitions, assumptions, and response mode are reviewed, the experts will be more consistent in their judgments (Meyer, 1986, p. 96). They will be more consistent between and within themselves. In addition, if there is much time between this first review and when the experts' judgments are requested, the question can be worded to include some of the above information. For example, "What rating would you give to the importance of element *X* over *y* to the reaching of goal *z*?" If they are using a response mode, in this case a Saaty paired comparison, they will need to have the definitions of the scale available in front of them.

Another technique for reducing inconsistency is to have the group members monitor their own consistency (Meyer, 1986, p. 96). This technique was successfully used in a simple interactive group elicitation where the experts were able to watch the knowledge engineer's monitoring of inconsistency and then mimic it (Meyer, Peaslee, and Booker, 1982). The experts were given copies of a matrix of the elements being judged, the criteria on which they were being judged, and their past judgments. When experts monitor their own consistency they may wish to change an earlier judgment to be in line with their current thinking. If their reasoning does not violate the logic of the model or the definitions, they can be allowed to make the change. Often in this process, the expert may discover that he had forgotten to include some pertinent information. After this addition, some of the judgments may need to be redone.

If Saaty's Analytic Hierarchy Process (1980) had been used and its results indicated high inconsistency, the experts could review and redo the affected judgments.

Availability. Availability bias can be countered by stimulating the expert's memory associations. In general, group discussion will cause the expert to think of more than just the first readily accessible information. In addition, free association can be introduced to single experts or those in groups. Free association is having the expert or experts quickly generate any and all elements that might have bearing on the question (Meyer, 1986, p. 94). Free association is similar to brainstorming or the Crawford Slip method (Boose and Gaines, 1988, p. 38). The experts are asked to refrain from being critical in order to generate the widest possible pool of ideas. (This list is later narrowed to those judged to be most pertinent to the problem.) A related technique is to hierarchically structure the presentation of problem information so that it flows from the general to the specific. In this way, the expert is able to consider the pertinent information before having to reach a solution. Again this strategy is to fire as many memory associations as possible so that the maximum number of relevant ones will enter into the expert's final judgment.

Anchoring. Techniques similar to those used to counter availability bias are used to counter anchoring. In particular, giving the expert input from other experts as in a Delphi situation or an interactive group makes it more difficult for the expert to anchor to his first impression. Another technique is to ask the expert for extreme judgments before getting his likely ones (Cleaves, 1986, p. 9-10).

2.5. Step 5--Analyzing the Data for the Occurrence of Bias

This step can be conducted at two different levels: (1) simple analyses and (2) complex analyses.

2.5.1. Simple Analyses. The first level is often used if step 5 is the only one that the knowledge engineer will be doing. In addition, these simpler tests are frequently used on cognitive biases because they are more amenable to straightforward analyses.

However, even doing this simpler level of analysis, presupposes that sufficient data was gathered. One of the major problems in testing for bias is having a sufficient sample size. The sample can be composed of multiple experts' data on the same problem or a single expert's data on many problems. To ensure that there is enough analyzable data, it is helpful to consult with a statistician before beginning the elicitation. In addition, we would caution the analyst to beware of misrepresenting the data in analyzing it. The use of many statistical techniques require that the analyst make particular assumptions about the data. If these assumptions are not warranted, the data will be misrepresented or biased according to the motivational bias. For example, many statistical techniques require that the analyst assume that the data is normally distributed. The use of these techniques would often misrepresent the data, which in our experience has been multimodally distributed. For this reason, we recommend the use of nonparametric statistical procedures and data-based simulation techniques wherever possible because they do not require distribution assumptions.

Two examples of testing at a simple level are given below.

We tested for the presence of **group think bias** in a project whose goal was to elicit estimates on the relevance of selected weapon-related components to particular military needs (Meyer, Peaslee, and Booker, 1982). The experts had assigned weights to describe how a weapon would meet a defense need. The weights ranged from 0, meaning not at all related, to 3, meaning 80% to completely related. Group think had earlier been judged a possibility because the experts were from the same work group, had worked together before, and had been selected for this project by their supervisor, who was present to lead some of the sessions. In addition, behavioral aggregation was being used, which further predisposed the elicitation toward group think. Thus, it was possible that the leader's views were influencing the discussions and the assignment of weights. To test for this possibility, the estimates were grouped according to who had led the session and analyzed for correlations to the weights. There was no evidence of this effect, but the conclusion was not absolute because some of the factors overlapped making definitive testing of their effects impossible.

On another project, the **underestimation of uncertainty** was tested for. The experts had estimated the likelihood of achieving national magnetic fusion milestones within particular time periods. This bias was considered likely because people typically underestimate uncertainty, and therefore, the time and money needed to complete various tasks. In addition, people seem to be more prone to this underestimation when they are estimating on uncertain situations, such as fusion technology. Research by Rand Corporation has shown that cost estimates for a first-of-a-kind project display greater cost errors than for other kinds of projects (Merrow, Chapel, and Worthing, 1979). For these reasons, the expert's ranges on a question were analyzed. They were found to be within one standard deviation of the data. This indicates that the experts thought that they were adequately accounting for uncertainty when they were only accounting for about 60% (Meyer, Booker, Cullingford, and Peaslee, 1982).

2.5.2. Complex Analyses. The second level of analysis involves using the data gathered from monitoring and adjusting for bias (steps 3 and 4) to do a more complete analysis. Information gathered during these steps, such as who appeared to be exhibiting a particular bias or

differed in their problem-solving processes, can offer insights into what biases to test for. In addition, if the later analysis yields confusing results, this data can aid in their interpretation.

For conducting the more complex analyses, we recommend the use of multivariate and simulation techniques. Multivariate techniques allow the simultaneous consideration of two or more variables of interest (Tietjen, 1986). This capability is important because expert data is frequently conditioned on other data. An expert's data can be conditioned on such factors as aspects of the elicitation situation and the expert's interpretation of the question and method of solving the problem. Because of the small sample sizes usually associated with knowledge acquisition, these effects can overlap and cause problems in the analysis. For example, in the early project mentioned above, we were not able to definitively test for group think because we could not separate this variable from others.

We have used several multivariate techniques, such as cluster analysis, discriminant analysis, and general linear models in trying to find the one most applicable to the data in question. If these techniques require distribution assumptions that are not applicable to the data, simulation techniques can be used. Like multivariate techniques, simulation techniques can be used to discover relationships between the variables; unlike them, they do not impose assumptions about the distributions. A statistician can provide guidance in selecting the specific analytic techniques.

Two examples of tests performed at this more complex level are given below.

In two studies we searched for sources of correlation, or dependence, between expert's solutions (Booker and Meyer, 1988; Meyer and Booker, 1987). The experts solved problems in individual interviews. Data was gathered on aspects of the expert's professional backgrounds, the problems, and the expert's means of solving the problem. This data was combined in a large data base, and its multivariate relationships were analyzed using multivariate and simulation techniques.

In another study, **anchoring bias** was examined using an experimental design. The experts in this project received limited information from individuals threatening violence. They had to decide if the communication was a ploy or if it should be taken seriously. During the monitoring of the first set of elicitations (step 3), the knowledge engineer discovered that the experts were making implicit assumptions. Follow-up interviews indicated that about half of the experts assumed that the communications were ploys until proven otherwise; the other half assumed that they were credible until proven otherwise. For the problem-solving sessions, experts were assigned to questions so a pair of each assumption type would be represented by a problem. With such a design and the earlier mentioned techniques, it was possible to test for the effect of this assumption on the expert's final conclusion.

3. SUMMARY

A preliminary program for treating bias has been proposed here. Its steps are (1) anticipating which biases are likely to occur in the planned elicitation; (2) making the knowledge engineer and expert aware of the potential for these biases; (3) monitoring the elicitation for their occurrence; (4) adjusting, in real time, to counter their occurrence; and (5) analyzing the data to determine more definitively if they occurred. Its steps can be followed in sequence or singly, depending on the needs of the knowledge acquisition task.

While this program was developed for manual knowledge acquisition, it could be applied in part to automated, or interactive, acquisition. In automated knowledge acquisition, there will still be a need for the knowledge engineer, according to Boose and Gaines (1988, p. 11). Thus, the knowledge engineer could, in between assisting experts with the automated tools, follow the steps

for countering bias. Furthermore, the automated tools could assist the knowledge engineer in detecting and countering bias, especially those covered by the cognitive definition.

In this paper, we aim to provide a more disciplined approach to countering bias in elicitation. Such an approach benefits not only manual knowledge acquisition but creates a foundation for automating the countering of bias. Through this paper, we hope to stimulate other work on handling bias in knowledge acquisition and to obtain feedback for developing our proposed program.

The information in this paper is excerpted from chapters 1, 7, 8 , 9, and 10 of the draft book *Eliciting and Analyzing Expert Judgment: A Practical Guide (NUREG/CR-5424).*

The financial support of the Nuclear Regulatory Commission (FIN #7225-5) for the writing of the book is gratefully acknowledged. Thanks also go to Wilma Bunker, the editor of the book and of this paper.

REFERENCES

Armstrong, J. S. (1981). Long-Range Forecasting: From Crystal Ball to Computer, New York, NY: Wiley-Interscience.

Ascher, W. (1978). Forecasting: An Appraisal for Policymakers and Planners, Baltimore, MD: John Hopkins University Press.

Baron, R. A., and Bryne, D. (1981). Social Psychology: Understanding Human Interaction, Boston, MA: Allyn and Bacon.

Benbasat, I., and Dhaliwal, J. S. (1989). A Framework for the Validation of Knowledge Acquisition, *Knowledge Acquisition* , Vol.1, no. 2, p. 215-233 .

Booker, J. M., and Meyer, M. A. (1988). Sources and Effects of Interexpert Correlation: An Empirical Study, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 18, no. 1, p. 135-142.

Boose J. H., and Gaines, B. R. (1988). Knowledge Acquisition For Knowledge-based Systems, *Tutorial MP4 of the Seventh National Conference on Artificial Intelligence of the American Association for Artificial Intelligence,* St. Paul, MN, August .

Capen, E. C. (1975). The Difficulty of Assessing Uncertainty, presented at the Society of Petroleum Engineers AIME 50th Annual Fall Technical Conference, Dallas, Texas, September .

Clancy, W. J. (1989). *IEEE Expert.*, Vol. 4, no. 2, p. 9-24.

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, p. 9-0 to 9-11.

Dawes, R. M., Faust, D., and Meehl, P. E. (1989). Clinical Versus Acturial Judgment, *Science*, Vol. 243, p. 1668-1673.

Denning, P. J. (1986). The Science of Computing: Will Machines Ever Think? *American Scientist,* Vol. 74, p. 344-346.

Faust, D., and Ziskin, J. (1988). The Expert Witness in Psychology and Psychiatry, *Science*, Vol. 241, p. 31-35.

Festinger, L. (1957). A Theory of Cognitive Dissonance, Palo Alto, CA: Standford University Press.

Gaines, B. R. (1989). *Knowledge Acquisition,* Vol. 1, no. 1, p. 39-58.

Goffman, E. (1959). The Presenation of Self in Everyday Life, Garden City, NY: Doubleday Anchor Books.

Gold, P. E. (1987). Sweet memories, *American Scientist*, Vol. 75, 151-155.

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, p. 471-476, Los Altos, CA: Morgan Kaufman Publishers.

Hogarth, R. M. (1980). *Judgement and Choice*, New York: Wiley and Sons.

Janis, I. C. (1972). Victims of Group Think: A Psychological Study of Foreign Policy Decisions and Fiascos, Boston, MA: Houghton Mifflin.

Martz, H. F., Bryson, M. C., and Waller, R. A. (1985). Eliciting and Aggregating Subjective Judgments--Some Experimental Results, in the *Proceedings of the 1984 Statistical Symposium on National Energy Issues*, Nuclear Regulatory Commission report NUREG/CP-0063, p. 63-82, Seattle, WA.

McGraw, K. L., and Harbison-Briggs, K. (1989). Knowledge Acquisition: Principles and Guidelines, Englewood Cliffs, NJ: Prentice-Hall.

Merrow, E. W., Chapel, S. W., and Worthing. C. (1979). A Review of Cost Estimation in New Technologies: Implications for Energy Process Plant, Rand report R-2481-DOE, Santa Monica, CA.

Meyer, M. A. (1986). Human Factors Affecting Subjective Judgments, in the *Proceedings of the Thirty-First Conference on the Design of Experiments in Army Research Development and Testing*, p. 85-98, Madison, WI.

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, Los Alamos, NM.

Meyer, M. A., Booker, J. M., Cullingford, H. S., and Peaslee, A. T., Jr. (1982). 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, p. 421-430, Ann Arbor, MI: Ann Arbor Science.

Meyer, M. A., and Johnson, E. R., (1985). Proceedings from the Technology Control Panel Workshop, Los Alamos National Laboratory technical report LA-UR-85-4078, Los Alamos, NM.

Meyer, M. A. , Mniszewski, S. M., and Peaslee, A. T. Jr. (1989). Using Three Minimally Biasing Elicitation Techniques for Knowledge Acquisition, *Knowledge Acquisition*, Vol. 1, no. 1, p. 59-71.

Meyer, M. A., 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, Los Alamos, NM.

Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information, *Psychological Review*, Vol. 63, p. 81-97.

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.

Ortiz, N.R., Wheeler, T. A., Meyer, M.A., and Keeney, R. L. (1988). The Use of Expert Judgment in NUREG-1150, (SAND88-2253C), to appear in the Proceedings of the 16th Water Reactor Safety Information Meetings, October 1988, Gaithersburg, Maryland.

Payne, S. (1951). The Art of Asking Questions, Princeton, NJ: Princeton University Press.

Rosenfield, I. (1988). The Invention of Memory: A New View of the Brain, New York: Basic Books.

Saaty, T. L. (1980). The Analytic Hierarchy Process: Planning, Priority Setting, and Resource Allocation, New York, NY: McGraw-Hill.

Seaver, D. A. (1976). Assessments of Group Preferences and Group Uncertainty for Decision Making, Social Science Research Institute, University of Sourthern California, Los Angeles, CA.

Spetzler, C. S., and Stael von Holstein, C-A. S. (1975). Probability Encoding in Decision Analysis, *Journal of the Institute of Management Science*, Vol. 22, p. 340-358.

Tietjen, G. L (1986). A Topical Dictionary of Statistics, New York: Chapman and Hall.

Weissenberg, P. (1971). Introduction to Organizational Behavior: A Behavioral Science Approach to Understanding Organizations, Scranton, OH: Intext Educational.

Zimbargo, P. G. (1983). To Control a Mind, *Standford Magazine*, Winter, p. 59-64.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

HYPERMEDIA IN KNOWLEDGE ACQUISITION AND SPECIFICATION OF USER INTERFACE FOR KBS: An approach and a case study

Jukka Rantanen
Laboratory for Information Processing
Technical Research Centre of Finland (VTT)
Lehtisaarentie 2 A
SF-00340 Helsinki, Finland

e-mail: Rantanen_J@FINVTT.BITNET

ABSTRACT

This paper describes the use of the hypermedia systems and tools in the knowledge acquisition and in the specification of user interfaces for knowledge-based systems. A pragmatic approach that combines informal task analysis with object-oriented modeling is introduced. This is supported by a hypermedia-based specification and documentation tool for knowledge requirements and user interface requirements. This approach and the tool are illustrated with examples from a case study: a specification of an intelligent user interface for a statistical analysis package.

1. INTRODUCTION

The hypermedia and hypertext may be seen as a new approach to the development of expert systems and knowledge-based systems (KBS). The KBS knowledge bases may be considered as a kind of hypermedia and hypertext systems, that consist of nodes (concepts, facts, texts, graphics, sounds) interconnected with static and dynamic links (relations, rules). The techniques and tools of hypermedia and hypertext are improving and emphasizing the capabilities of a KBS in **communication and structuring of domain knowledge**, that are important both for routine use, knowledge acquisition, and maintenance of a KBS. This paper is talking only about the hypermedia, because combination of text and graphics will be increasingly important in the KBS applications; the term hypermedia is used here to refer also to the hypertext. The capabilities of hypertext and hypermedia have been explored in research institutes, but recent introductions of these systems and their authoring tools for personal computers have made them widely feasible in various applications (Cocklin 1987).

The **acquisition of knowledge** has been often seen as a major bottleneck in the building of KBSs, and not infrequently it is accomplished with **ad hoc approach**. In practice, sketchy specifications and documentations of knowledge (requirements) have been almost a rule, and working prototypes have been in the role of the major formal detailed specifications. Recently some promising applications of hypermedia tools and systems in the knowledge acquisition, analysis and documentation have been appearing.

The development of easy-to-use **user interfaces** has also gained much attention in literature, but the practice of development seem to be dominated more by **ad hoc approaches** than by the application of the many written guidelines and good principles. A collection of good examples are often the major guideline (e.g. Hendler 1988). Experiences from the practice tell that only when there is an easy-to-use tool available for rapid prototyping of user interfaces, the

good principles of rapid prototyping and user evaluation will work. Otherwise these principles are not followed in practice. Here again some of the new hypermedia tools seem to provide a promising vehicle that combines rapid specification of a user interface to rapid implementation and evaluation of its prototype.

This paper shall first consider the applications of hypermedia techniques and tools in knowledge acquisition, analysis and documentation in relation to the methodologies for the development of KBSs. Then it will be introduced a pragmatic approach to knowledge acquisition and the specification of a user interface, that combines an informal task analysis with an object-oriented conceptual modeling. It is based on more extensive guidelines (Rantanen 1988). The approach is supported by a hypermedia-based tool for knowledge requirements specification and documentation. The application of the proposed approach and the tool is illustrated by a case study. In this case the specification of knowledge, the user interface requirements and the working implementation did coexist in the same dynamic prototype that was rapidly implemented by the HyperCard tool. The intelligent user interface was made for a statistical package that was a part of an oncological decision support system for selection of cancer therapy (Hyödynmaa et. al. 1988).

2. USES OF HYPERMEDIA IN KNOWLEDGE ACQUISITION AND ANALYSIS

The knowledge acquisition and analysis is a major task in the development of a KBS, the result of which is called a knowledge model or a problem solving model. This task may also be characterized as the specification and analysis of knowledge requirements, that are themselves related to the conventional specifications of functional, data and user requirements. In the following is a short description of the different levels and viewpoints of requirements specification for a KBS (Fig. 1 and more details in the appendix):

- application domain level view : domain, task and user requirements specification
- system level view: functional and data specification
- knowledge level view: specification of the preliminary knowledge requirements and the resulting knowledge model
- implementation level and view: KBS architecture and design specification.

The domain and the task are here conceptualized and modeled from the different perspectives of domain experts, end-users, system analysts, and knowledge engineers. who are using different terms and techniques. In the rapid prototyping of demonstrative KBSs the specifications have often been made only on the application and implementation level. However, in the development of more generic KBSs there is the demand for the more abstract and formal specifications, that are made in the system and knowledge level.

In the following the uses of hypermedia are related to the four levels and views of specification. Hypermedia tools may be used to implement dynamic specification documents that represent these multiple views and mappings between them, and allow stepwise refinement of rough informal specifications to detailed formal specifications.

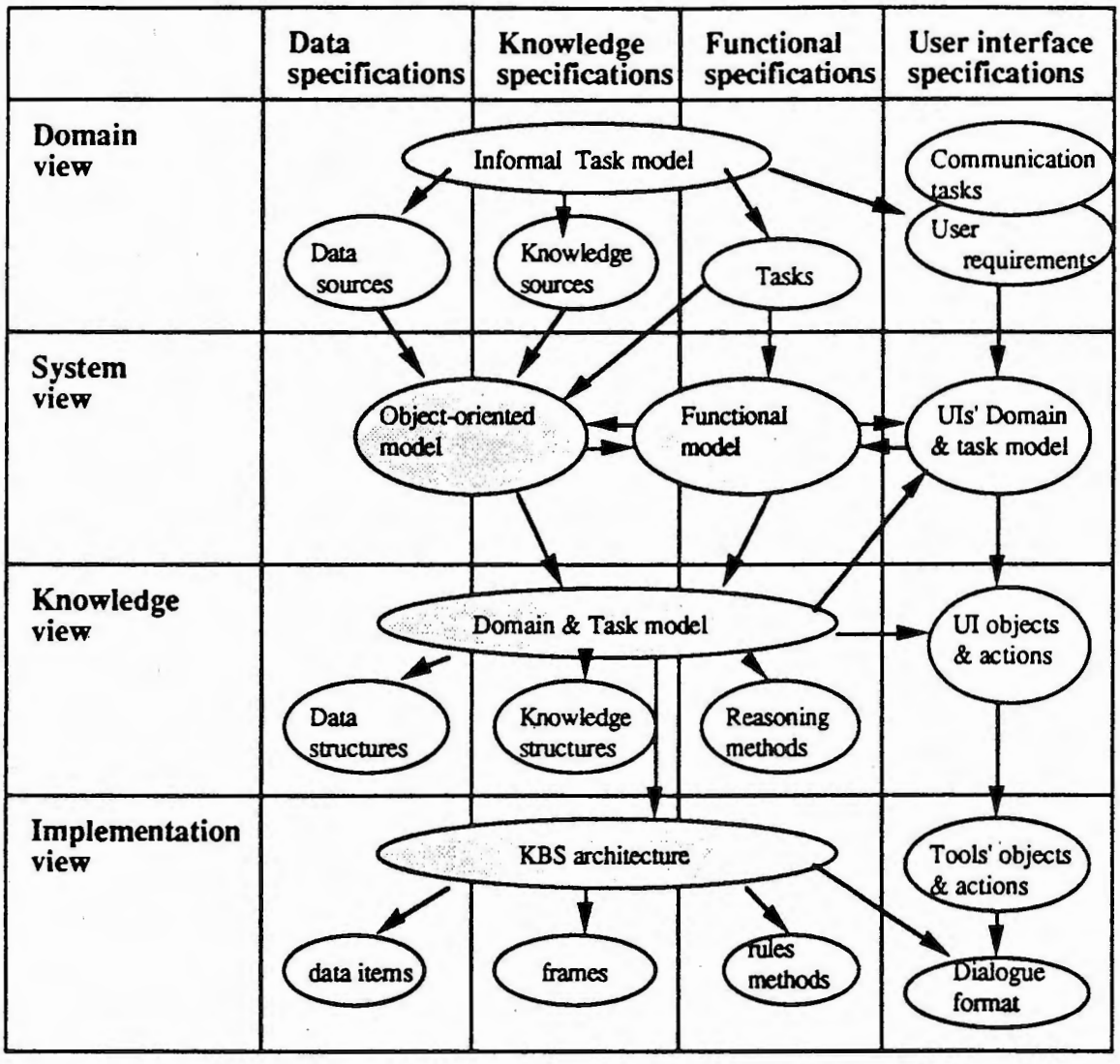


Figure 1. The different levels and viewpoints of the requirements specification of a KBS.

2.1 Domain View and Domain Analysis

In the informal domain analysis domain experts are describing and structuring the problem or the task by their own domain vocabulary and illustrations. The aim of the analysis is first to decompose the task into smaller subtasks or subproblems, to identify relevant knowledge subdomains and sources, and then to specify the inputs, outputs and solutions of the problem. This domain analysis consists of

- analysis and specification of the problem solving task and the major phases of the task-related problem solving,
- analysis and specification of the communication task, including the user interface, a consultation strategy, and the explanation capabilities, and
- specification of other requirements of the work environment.

In the **informal domain analysis** one is using informal knowledge acquisition, analysis, and documentation techniques. It is especially here where **hypermedia-based**

documentation systems may be of great help in bridging the gap between preliminary outlines and the more formal and detailed specifications. On the other hand a hypermedia-based prototype may quickly illustrate and make concrete the requirements and possibilities for domain experts, and on the other hand a hypermedia-based documentation forces the knowledge engineer to specify the requirements from the user interface point of view. Another advantage is that the specification documents may be easily tailored to meet the needs of a specific domain, a task or a project. Moreover, the documentation can be minimal or extensive depending of the purpose of the project.

2.2 System View and System Analysis

System analysts are describing and analyzing the problem or the task by the formal specification techniques and by using the vocabulary of system analysis. The task may be analysed and decomposed into functions or data objects:

- the functional specification approach: the specification of functional and data requirements are usually done by the structured analysis (SA) technique, or
- the conceptual modeling approach: the domain or the task is modeled by object-oriented conceptual modeling that leads to specification of data and knowledge.

The hypermedia tools and documentation systems may be used for the specification of functional and data requirements. For example the structured analysis method has been implemented by the HyperCard tool (Priha 1988). These implementations are only showing the potential, they are not as sophisticated and efficient as the CASE (Computer Aided Software Engineering) tools.

Some hypermedia tools (e.g. HyperCard) support object-oriented modeling that is made concrete by user interface facilities (metaphors, icons, graphics). With these tools the bridge from the abstract object-oriented conceptual modeling to a working object-oriented implementation is made more direct and clear. This is an important achievement, because in the past there has often been a big difference between a conceptual model and its implementation. The hypermedia tools shall be even more and more useful for object-oriented conceptual modeling, when they shall include a better integration of the graphical and textual facilities, so that it is easy to go from the diagrams to the corresponding textual specifications. One may expect that the useful ideas of the hypermedia documentation systems and object-oriented modeling shall be integrated in the forthcoming CASE tools.

2.3 Knowledge View and Knowledge Analysis

Knowledge engineers in co-operation with domain experts are analysing and making more refined specifications of the knowledge and problem solving methods that are required in the task. This consists of the preliminary specification of knowledge requirements followed by knowledge acquisition and analysis, the approach of which may be

- **transcript data -oriented**: in the beginning nobody has clear view of the required knowledge or problem solving methods, that are discovered little by little during the knowledge acquisition, analysis and prototyping cycles.
- **model -oriented**: the knowledge requirements are preliminary specified by appropriate problem-solving models or by more concrete KBS architectures.

The specification of knowledge requirements starts usually from the making of an overview of the problem solving task and the knowledge subdomains by the **informal task analysis**, that is done by DEs in the informal domain analysis. This is followed by the preliminary

structuring of the knowledge or problem solving model, which is done by applying appropriate knowledge classifications, problem-solving models and methods, as well as ideas from relevant reference systems. Indeed, the discovery of the preliminary knowledge structure is an important task for knowledge engineers, who should make a survey and analysis of the relevant problem-solving models, architectures, and reference KBSs. This helps to overcome the outstanding problem that is often met in the beginning of the knowledge acquisition task: domain experts have an overwhelming amount of knowledge, but they feel incapable to find any structure for it.

Hypermedia tools may be used in many ways in the specification of knowledge requirements, knowledge acquisition, analysis and documentation:

- **a documentation system for knowledge requirements**, that support also the preliminary specification before acquisition. More capable tool could support the preliminary structuring of the knowledge or problem solving model, which might include reference links to the relevant models and KBSs, and also to the short descriptions and analysis texts that are results of this kind of reference survey.

- **a knowledge acquisition and analysis tool**, that is based on a problem-solving model that is appropriate to the task at hand. This emphasizes the usefulness of the initial specification and modeling of the knowledge requirements, too. When the preliminary specification has been made properly and successfully, it becomes possible to develop a hypermedia tool for the knowledge acquisition and documentation tasks.

- **a knowledge acquisition and structuring tool for the domain expert**, that allows the expert himself to express, structure, and document his knowledge with the hypermedia tool or documentation system available for him. For instance, in those applications where the user interface and explanation capabilities are important, the domain expert can himself author the explanations and help texts with a hypermedia tool. These tools may be used also in the interviewing sessions for immediate documentation of the answers, explanations, and comments, if a successful structuring of knowledge has already been made. All these uses presuppose that a more general hypermedia tool is tailored to a problem-oriented system for specification documentation, which as well as possible supports structuring of knowledge that is appropriate to the task at hand.

- **a knowledge analysis and documentation tool**, that is used by knowledge engineers. The hypermedia tools may be used in the transcript analysis of application documents or literature. In the **transcription analysis** important terms, concepts, facts, principles, topics etc. are identified from the transcript data, which leads to an initial structuring of transcript data.

The transcript analysis is further refined and structured by **conceptual modeling**, where the conceptual structures are identified and specified. Object-oriented conceptual modeling from knowledge engineering point of view is close to that of system analysis view, because it is often difficult to make any difference between data and knowledge objects. Object concepts are often in the preliminary analysis chunks of data and knowledge, and the object specification is a useful overview without any need for further differentiations.

The **knowledge level analysis** is the final more abstract analysis and specification of knowledge and problem-solving methods. The transcript data or the results of the transcript analysis or object-oriented modeling are further analysed and specified by application of appropriate problem-solving models and their analysis vocabulary (e.g. fault, indication, symptom, heuristic classification method etc.). This may be based on the KADS methodology or other less refined analyses of knowledge structures and functions (Wielinga & Breuker 1986, Hayward et. al. 1987, Clancey 1986).

Some hypermedia tools are suitable to the transcript analysis, because they support appropriate annotation techniques (e.g. Guide). With other tools it is easy to make specification forms for objects, methods, and rules in order to document the results of conceptual modeling (e.g. HyperCard). More sophisticated hypermedia-based tools shall be developed for the more advanced knowledge analysis methods (e.g. KADS), when more capable hypermedia tools are available.

2.4 Implementation View and Specification

Knowledge engineers or programmers specify and design the computer implementation of a KBS by applying suitable computerized AI techniques. This includes

- selection and specification of suitable KBS architecture
- selection of suitable knowledge representation techniques and representation of the knowledge
- selection of problem-solving techniques and design of the problem solving mechanism.

In many practical projects the rapid prototyping approach and the adopted KBS shell has guided the implementation so that the principle design phase has been neglected. Only later it has been found that the generic program architectures should have been considered already in the beginning of the KBS design phase. A recurrent problem is the lack of adequate requirements documentation, that is a reason to the inappropriate implementations (the throw-away prototypes). The requirements specification and documentation are not infrequently seen less necessary tasks. Especially when prototypes are rapidly made for demonstration purpose.

The hypermedia tools will change these practices, too. From object-oriented hypermedia-based specifications it is easy to move to object-oriented implementations, when a suitable tool are available. Some tools allow gradual transformation of specifications to a prototype implementation (e.g. HyperCard). Finally, the specification documents and the documentation of an implementation can meet each other. In this way the hypermedia tools are also improving the communication between the builders of conceptual models and the programmers, who are becoming more like authors than programmers of software.

3. A PRAGMATIC APPROACH TO KNOWLEDGE AND USER INTERFACE SPECIFICATION

A pragmatic approach to the knowledge acquisition and the specification of user interface shall be introduced now. It is based on more extensive guidelines, that are a synthesis of recent research on knowledge acquisition methodology (Rantanen 1988). The pragmatic approach is an attempt to combine the following three specifications

- informal task analysis (TA) with
- object-oriented conceptual modeling (OM) that may be further mapped to
- user interface specifications and prototyping .

In rapid prototyping the minimal, easy-to do specifications and documentation are favoured, but often one has not made any adequate overview and structuring of the problem. In practise it is often difficult to start an extensive knowledge acquisition and prototype development without

making an overview and initial structuring, or a sketchy specification, of the task, of the required knowledge and of the user interface. The proposed pragmatic approach tries to meet these needs, first by starting from a collection of informal preliminary specifications from which one may then go on to more formal specification techniques or to the more detailed specifications that may be dynamically prototyped by a hypermedia tool (Fig. 2).

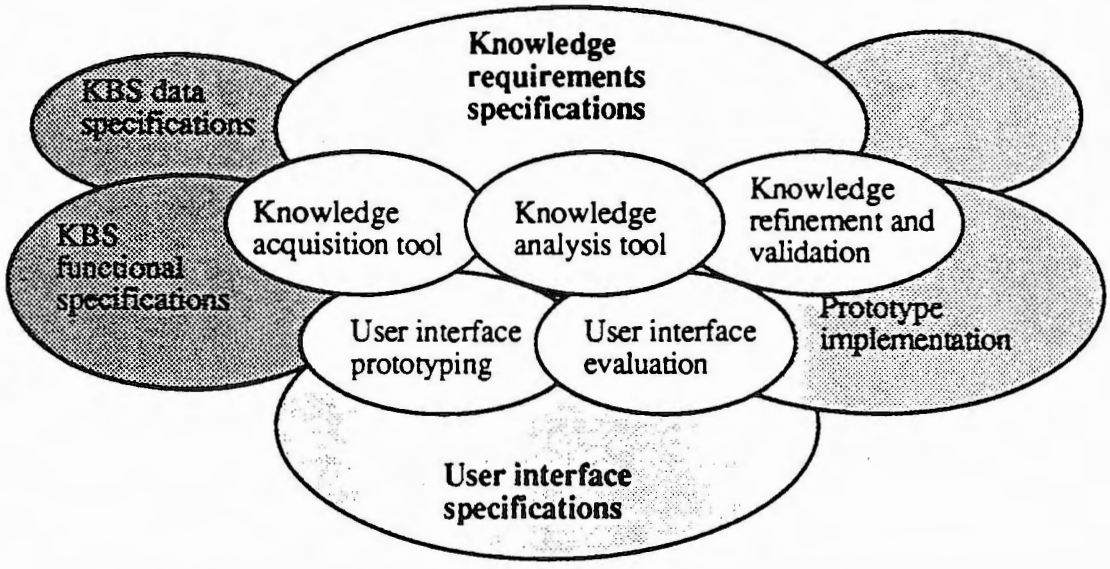


Figure 2. The dynamic specification and rapid prototyping of knowledge, data, functional and user interface requirements with hypermedia tools.

3.1 Knowledge Acquisition and Requirements Specification Cycles

The specification of knowledge and user interface requirements may be divided into the following main phases (Fig. 3):

1) problem overviewing cycle and informal domain analysis: in the first informal domain analysis domain experts are outlining and making overview of the domain and problem. Here one may use for example the wall chart technique in the outlining of the first sketchy specification results of the informal task analysis and the preliminary object-oriented modeling. After the task decomposition and object identification has been made, preliminary screen layouts may be done. The results of this domain analysis is documented by the hypermedia forms for specification of the domain knowledge requirements together with overview diagrams.

2) knowledge structuring cycle and requirements specification: the preliminary structuring of domain and task made by domain experts is further analysed and specified by knowledge engineers. The preliminary structuring of the knowledge of the domain and task is attempted by applying relevant knowledge classifications, problem solving methods and models. The knowledge and user interface requirements are documented by a hypermedia tool. This is the second hypermedia-based specification cycle that is aiming to an acceptable or to a more generic and formal structuring of the knowledge and problem solving methods required by the task.

3) refinement and validation cycle with knowledge acquisition and analysis: a successful structuring of the knowledge (requirements) may provide bases for using a hypermedia tool in knowledge acquisition. Often it may be used only for the documentation of knowledge specifications, when knowledge acquisition is done by such usual techniques as interviewing. This is the third hypermedia-based specification cycle, where knowledge acquisition and analysis is made using a hypermedia tool.

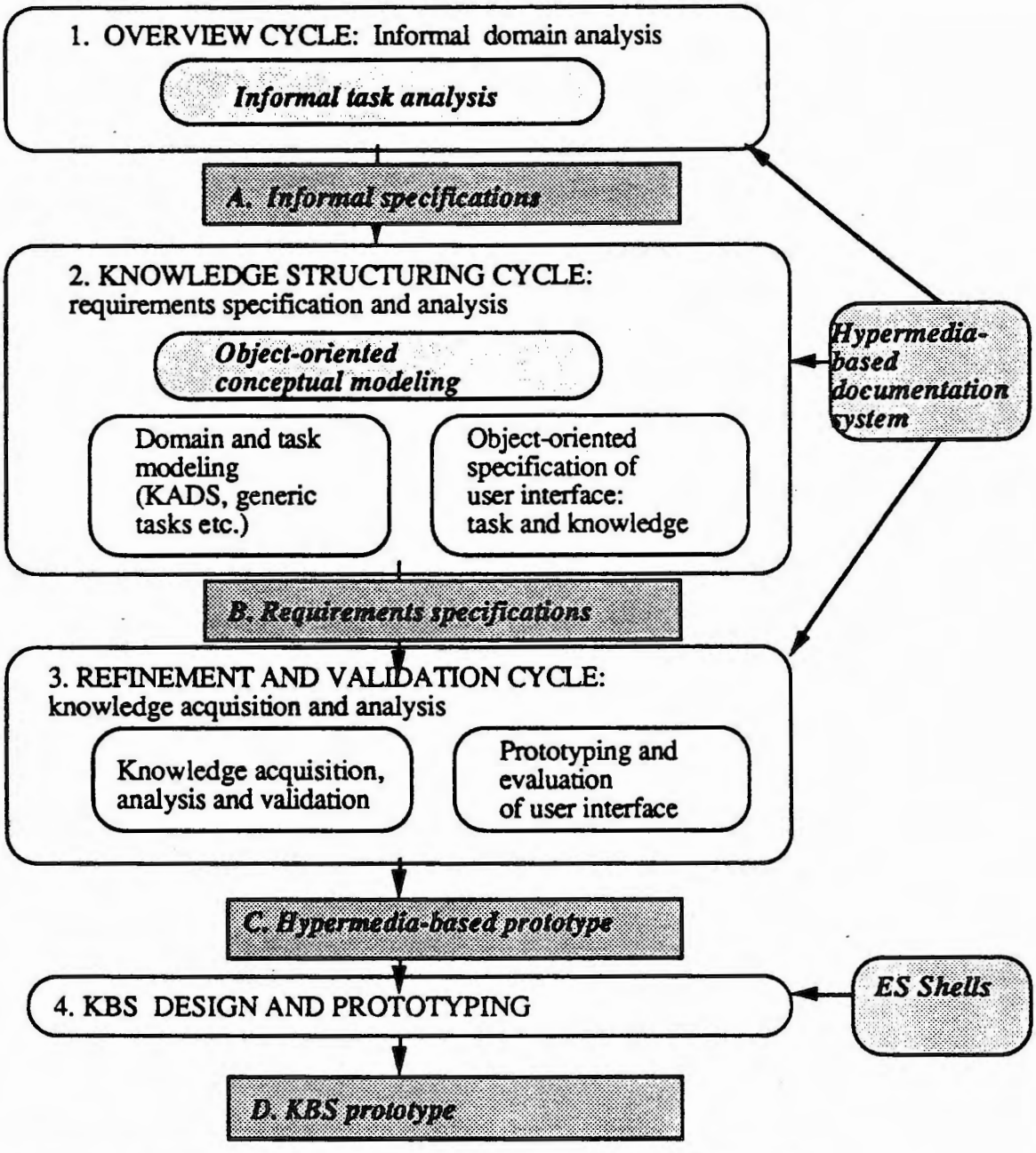


Figure 3. A pragmatic approach to the specification of knowledge and user interface requirements.

The initial structuring and the informal specification of knowledge, data and functional requirements are made by the informal task analysis followed by the object-oriented modeling.

The **informal task analysis** is an easy way to start the requirements specification from the application domain point of view. It is a result of the domain analysis and a minimal documentation for it. The results of informal task analysis may be further refined and abstracted by the KADS-methodology (Wielinga & Breuker 1986) or by identification and specification of the generic problem solving tasks (Chandrasekaran 1987) or by the extended task analysis (e.g. Keravnou & Johnson 1986).

The **object-oriented conceptual modeling** is an important and useful next step in the specification, because no difference between data and knowledge is not yet made. It brings into focus the access to data and knowledge sources, which is crucial in many practical applications. It provides also a useful overview of knowledge structure and an abstraction of the problem at hand. From object-oriented model one may proceed to functional specification of the system and to the knowledge requirements specification that includes more general classifications of knowledge, problem solving models and methods. Examples of task decompostion diagrams and object descriptions will be presented in the case study below.

3.2 Knowledge-based User Interface Specification

The informal task analysis and object-oriented conceptual modeling are first steps to more detailed **specifications of user interface**. The user interface specification includes specification of the contents and the format of the dialogue between the user and the system. The **object-oriented and the knowledge-based specification and design** approaches are more and more applied in the design of the contents of dialogues. Adding intelligence to the user interface means that user interface is designed from the knowledge engineering point of view. This means that one should identify useful task-related knowledge and explanations that are provided on the display screens and windows. From the informal task model and object-oriented model it is possible to identify the task-related knowledge objects for the user interface. Subtask-related objects can often mapped into task-related displays (in Hypercard these are (display) cards). The preliminary specification of the user interface by a hypermedia tool allow further refinement of the content. It is also a kind of knowledge acquisition that is serving both the specification of the knowledge and the user interface. For instance, the explanation types and texts may be easily specified and evaluated by the hypermedia tools.

The hypermedia gives new ideas for the format design of the user interface, too. Hypermedia tools allow rapid prototyping of the layouts of displays, that may be immediately evaluated by the end-users. Even the feedback from these user evaluations might be collected by a tailored hypermedia tool.

3.3 Hypermedia-based Documentation for Requirements

This approach is supported by a hypermedia-based tool for knowledge requirements specification and documentation, that support minimal or extensive documentation (HyperCard-based Knowledge Requirements Specification tool HKRS) (Fig. 4). The hypermedia tool can provide an environment, where the dynamic knowledge and user interface specifications may coexist together with prototype implementations. By adding more functionality to the hypermedia-based specifications leads to a functional implementation, a working prototype. This kind of minimal dynamic documentation system for knowledge requirements forces the knowledge engineer to tackle all requirements and to consider the relevancy of different issues and approaches to the KBS at hand.

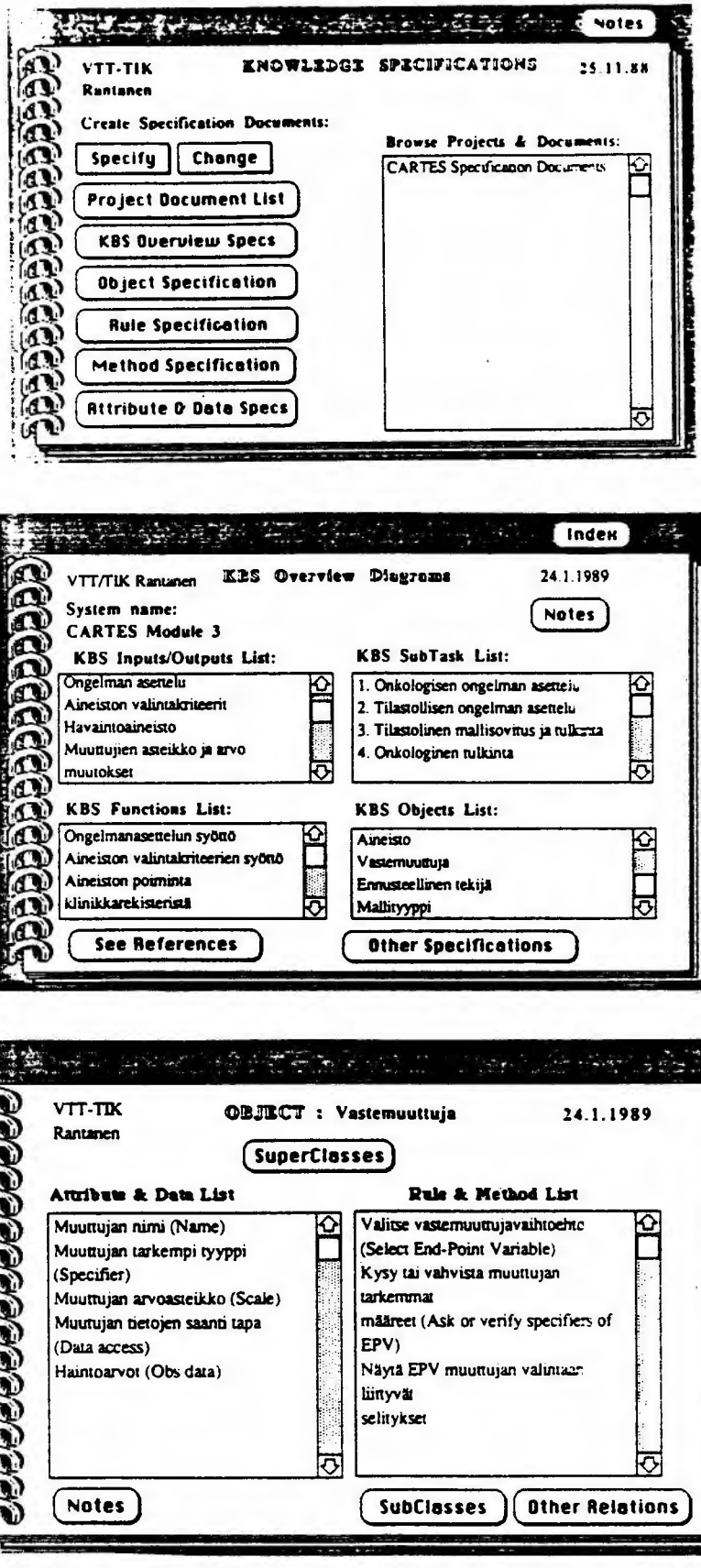


Figure 4. A hypermedia-based knowledge requirements specification system (HKRS): an example from the requirements specification of an intelligent interface (for the CARTES KBS).

The minimal documentation of the **informal domain analysis** phase consists of :

- task decomposition diagrams
- specification of input and output data
- specification of knowledge subdomains and sources.

The form layouts for the minimal specification documents can be easily authored by the HyperCard tool. Current limitation of this tool is the lack of direct linking from textual specifications (cards) to graphical descriptions (diagrams). So the specification texts and diagrams do not yet make up an integrated documentation system. But it seems that soon there will be available tools that support object-oriented graphics within a hypermedia-based documentation system (e.g. SuperCard).

In the proposed practical approach the **object-oriented conceptual modeling** is favoured as the system specification method. The object-oriented model is derived from the informal task diagram. The specification of the conceptual objects is first made by an overview diagram, which may may drawn by an appropriate tool (here MacDraw II or Design 2.0). Then each object may be further specified by identifying and naming

- the methods and reasoning rules for each object
- the data groups or parameters for each object and
- links to subobjects or superobjects.

From the specification of the objects of the application one may proceed to the functional specification or the specification of problem solving process.

The minimal system documentation is here done by the basic tools (HyperCard, MacDraw or Design), that allow tailoring of the specifications to the special needs of the project. There is not yet any generally accepted standards for object-oriented specifications, so this situation may lead to changing practices. But these rudimentary CASE tools are generally available and not so expensive as the sophisticated CASE tools.

The knowledge specifications include an overview part and object-oriented view of the domain and the task. The KBS overview is a checking list for major approaches and issues

- a listing of reference models and KBSs
- a listing of knowledge domains
- a listing of knowledge classification and organization principles, and
- a listing of generic problem-solving methods.

The object-oriented view includes specification cards for objects, rules, methods, and attributes with data.

4. A CASE STUDY OF HYPERMEDIA-BASED SPECIFICATION

The hypermedia tools seem to be very useful in the rapid prototyping and in the specification of knowledge requirements for communication-oriented systems, such as consultative expert systems or intelligent user interfaces. Of this a good example is an user interface module for an oncological decision support system (CARTES), that is assisting in the statistical analysis of a clinical register (Hyödynmaa et. al. 1988). A prototype of the intelligent user interface was developed in the end of the year 1988 to provide the following functionality:

1) oncological problem specification, that consists of oncological and statistical specification of the problem for statistical analysis. The starting point is the hypothesis about the relationship between the results of a treatment and the prognostic factors, which is

formulated by an oncologist. The intelligent user interface helps him to specify the problem and all the relevant data (variables, scales, and even oncological classifications that suggest the importance and weights of different prognostic factors) so that a certain statistical package may be used (in this case the package is the GLIM General Linear Models).

Usually an oncologist sends his problem specification and his data set to a statistican, who checks them, analyses them with a statistical package and then interprets the results of the statistical analysis, and finally writes answer to the oncologist.

2) statistical and oncological interpretation, that may be minimal or extensive explanation depending on the results of the analysis and the familiarity of the oncologists with the statistical analysis in question. The statistical interpretation may include comments and notifications, the background of which is not always clear to the oncologist. The oncological interpretation may lead to changes in treatments (protocols) and in treatment decision making; provided the associations between these results and the decision making rules are made more outstanding and structured.

After some general discussions about the task and domain, we found that best way to focus the interview sessions with the oncologist and the statistican was **case simulations**. They were preparing problem specifications and data sets for the package. This brought out a few communication issues, that cause problems in the conventional practice.

The major issue in this application seems to be the **clarification and disambiguation** of the communication between the oncologist and the statistican. Sometimes there seemed to be disagreements about some basic terms, positions and interpretations. It seemed to be important to make a clear difference between the oncological and statistical vocabulary, too.

Moreover, the users of statistical packages do not often know what should be included into a **good specification** of a problem for a statistical analysis and what extra information should be included in order to help the statistical interpretation. An intelligent user interface seems to be helpful in this case, because

- a routine user may briefly and quickly make his problem specification in a complete enough form for statistical analysis (variables, scales etc. may be provided readily for selection)
- a casual user may have on request more explanations that help him specify the problem and interpret the results.

The transcripts from the case simulations were analysed and structured by a informal task diagram, that served as an overview of the task flow and a framework for further interviews (Fig. 5). From the task diagram were identified major object concepts that were illustarted by task-object diagrams (Fig. 6). The object-oriented conceptual modeling and these informal diagrams were very helpful in the interviews with the domain experts. It seemed that they were first time considering more deeply the major concepts and their relationships, and there was the additional need to explicate them clearly to the others.

After having agreement of task decomposition and the major object concepts in the subtasks it was time to start to refine the knowledge specifications. Especially here the hypermedia-based documentation system were useful, because it allowed to browse through the task-object diagrams to fill the decriptions of objects, rules and methods. The user interface of the hypermedia tool was also helpful for the domain experts and the knowledge engineers. From the object-oriented model of the application task it was quite easy to proceed into the object-oriented implementation that was also done by the HyperCard tool. This prototype turn out to

be useful both for the further acquisition and refinement of knowledge and for the specification of the user interface.

The knowledge requirements of this module of the CARTES KBS is described with hypermedia documents and diagrams: the task decomposition diagram, the object-oriented conceptual model (Fig. 5 and Fig. 6), and examples of the specifications of some objects and rules are in the appendix (Fig. 2 and Fig. 3) .

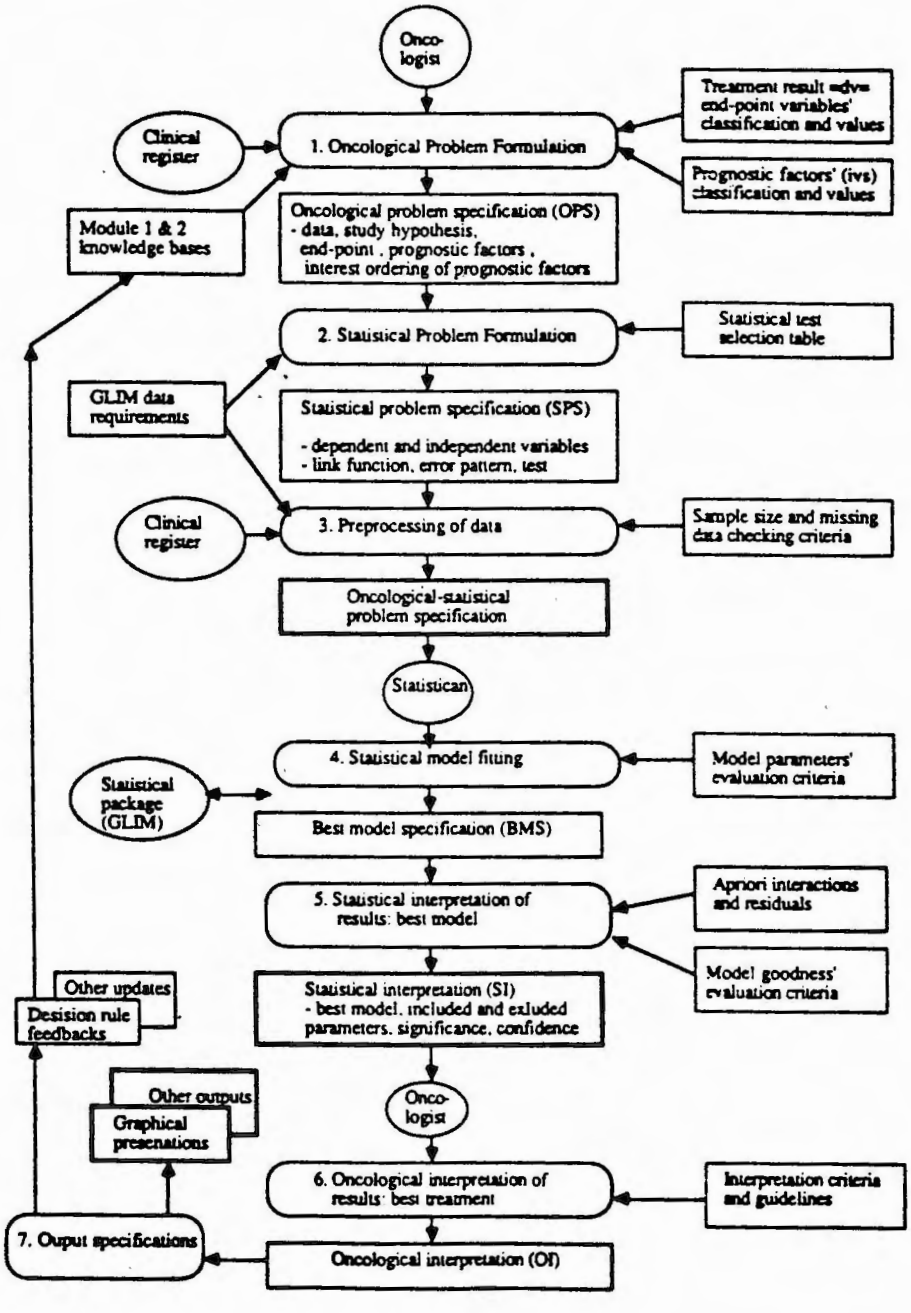


Figure 5. An example of informal task analysis (the task decomposition diagram) of an intelligent user interface for a statistical package (for the CARTES KBS).

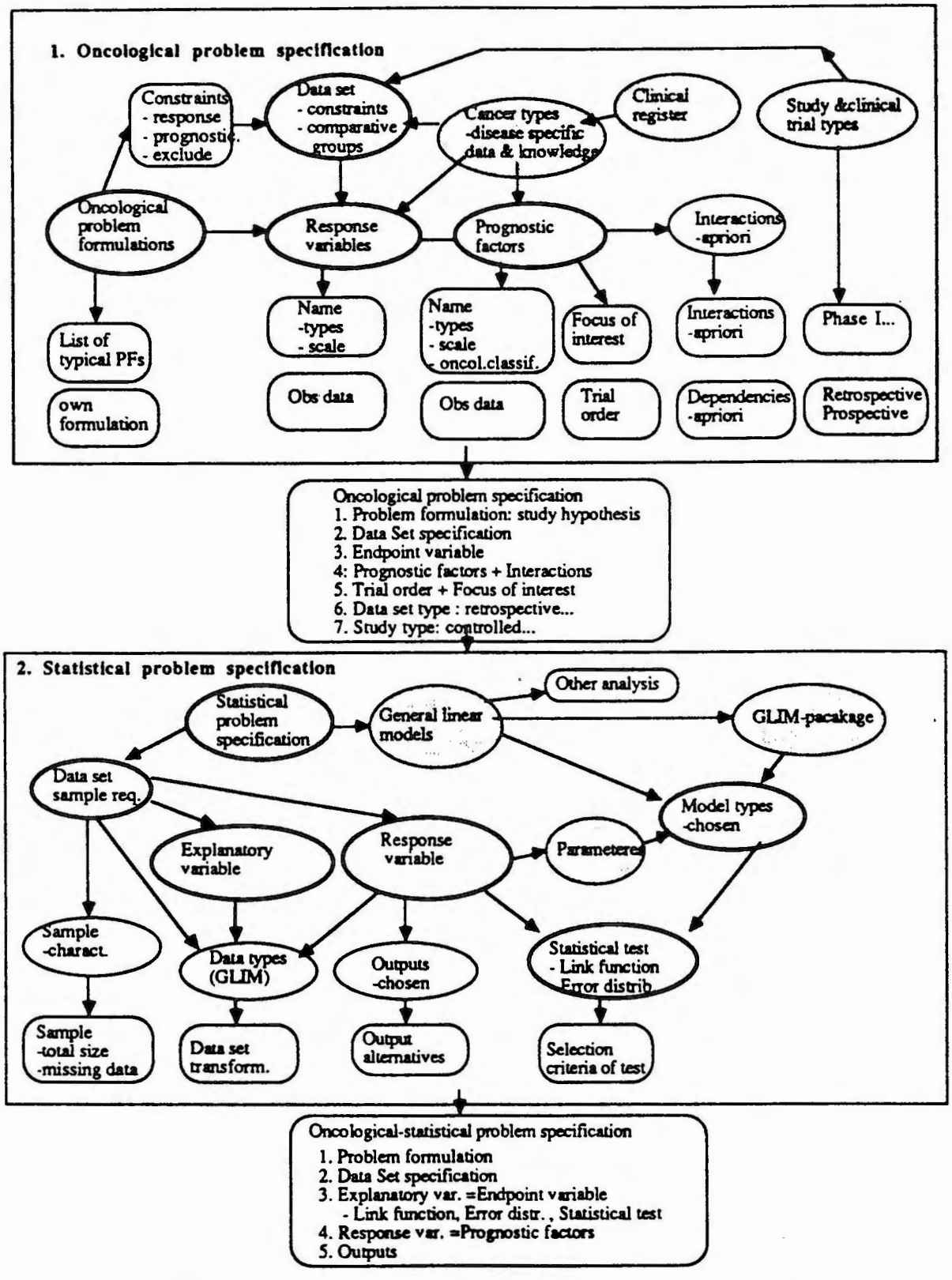


Figure 6. Identification of object concepts from the task decomposition diagram: an example of tasks and objects of an intelligent user interface for a statistical package.

5. CONCLUSION

The knowledge acquisition and analysis has been recurrently experienced as a major problem in the development of expert systems. A reason behind this problem has been inadequate user interfaces, that do not allow easy communication and structuring of knowledge. On the other hand in the development of intelligent user interfaces there is the need for knowledge-based and object-oriented specifications, that may be done easily and quickly. The hypermedia tools and documentation systems seem to provide to these problems a useful solution, that may be tailored to meet the needs of a specific application task or a project. In some cases they may be used even as knowledge acquisition tools that are used by the domain expert themselves.

In this paper a practical approach to the preliminary specification of knowledge and user interfaces was introduced. It was based on an informal task analysis from which an object-oriented conceptual model was derived. These specifications were further mapped into the objects of user interface. These preliminary specifications could be then further refined by hypermedia-based prototyping. For this purpose a minimal hypermedia-based system for the specification of knowledge requirements was developed (HKRS). This was an example of a system and solution that was unique to the application and the task at hand, but it could be implemented in a very short time by the available tool (HyperCard).

Finally, the use of this hypermedia-based specification system was illustrated by a case study of the prototyping of an intelligent user interface. The example shows how easily one may proceed from an object-oriented specification of the knowledge and the user interface to an object-oriented implementation, when an appropriate hypermedia tool is available.

The experiences from the use of the hypermedia-based specification tool (HKRS) tell that some improvements to the commercial hypermedia tools available are needed. They should support more directly object-oriented conceptual modeling and graphics, so that knowledge structures, text and graphics could be more easily integrated into the same system. When more experiences are acquired from the case-specific specification systems, such as the one described here, one may develop more generic, domain- or task-oriented hypermedia-based systems, that in many ways shall support knowledge requirements specification, acquisition and analysis.

A new generation of hypermedia tools are coming to the markets. They will be major tools for knowledge engineers, maybe more so than the current expert system shells. Because they are similar to the CASE-tools, one could call them knowledge-based multimedia tools or the CAKE (Computer Aided Knowledge Engineering) tools, that help to proceed from the specification of knowledge requirements, knowledge acquisition and analysis to the implementation of a prototype of a KBS.

ACKNOWLEDGEMENTS

This work has been funded by the Applications of Artificial Intelligence Programme of the Technical Research Centre of Finland (VTT).

REFERENCES

Chandrasekaran , B. (1987). Towards a Functional Architecture for Intelligence Based on Generic Information Processing Tasks. Proceedings of the IJCAI-87, Milano, pp.1183-1192.

Clancey , W. (1986). From GUIDON to Neomycin and Heracles in Twenty Short Lessons. AI Magazine, (7) 3, August 1986, pp.40-60.

Cocklin, J. (1987). Hypertext: A survey and introduction. IEEE Computer, September 1987, 20(9), pp. 17-41.

Hendler, J. (1988). (ed.) Expert Systems: The User Interface. NJ: Norwood. Ablex Publishing Co.

Hayward, S. , Wielinga, B. , Breuker, J. (1987). Structured analysis of knowledge. Int. J. Man-Machine Studies (1987) 26, pp. 487-498.

Hyödynmaa, S. , Näriäinen, K. , Ojala, A: , Rantanen, J. & Saranummi, N. (1988). Decision Support System in Oncology. In: Reichertz, P. & Lindberg, D. Lecture Notes in Medical Informatics. Berlin: Springer-Verlag. pp. 65-68.

Keravnou, E. , Johnson, L. (1986). Competent Expert Systems: a case study in fault diagnosis. New York: McGraw-Hill.

Kidd, A. (1987). Knowledge Acquisition: An Introductory Framework. In Kidd, A. (ed.). (1987). Knowledge Acquisition for Expert Systems: A Practical Handbook. New York: Plenum Press.

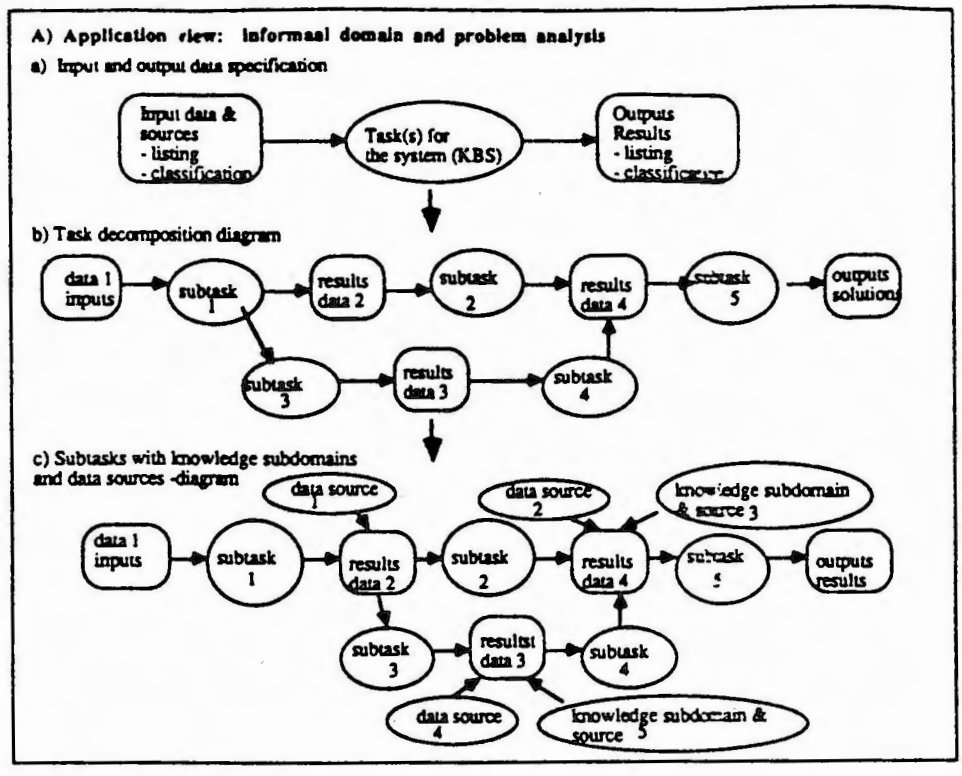
Priha, I. (1988). HyperCase (HyperCard-based implementation of the Structured Analysis technique). Technical Research Centre, Lab. for Information Processing.

Rantanen, J. (1988). Guidelines for Knowledge Acquisition and Documentation. In: Mäkelä, M. & Linnainmaa, S. & Ukkonen, E. (eds.) Proceedings of the STeP-88 (Finnish AI Symposium), pp. 576-582.

Wielinga, B. , Breuker, J. (1986). Models of Expertise. Proceedings of European Conference on AI (ECAI-86), Brighton. pp.306-318.

Hypermedia tools: HyperCard (Apple Computers, Inc. 1987), Guide 2.0 (Owl international, Inc, 1987), Design 2.0 (Meta Software, Corp.), MacDraw II (Claris, Corp.), SuperCard (Silicon Beach Software, Inc. 1989).

APPENDIX



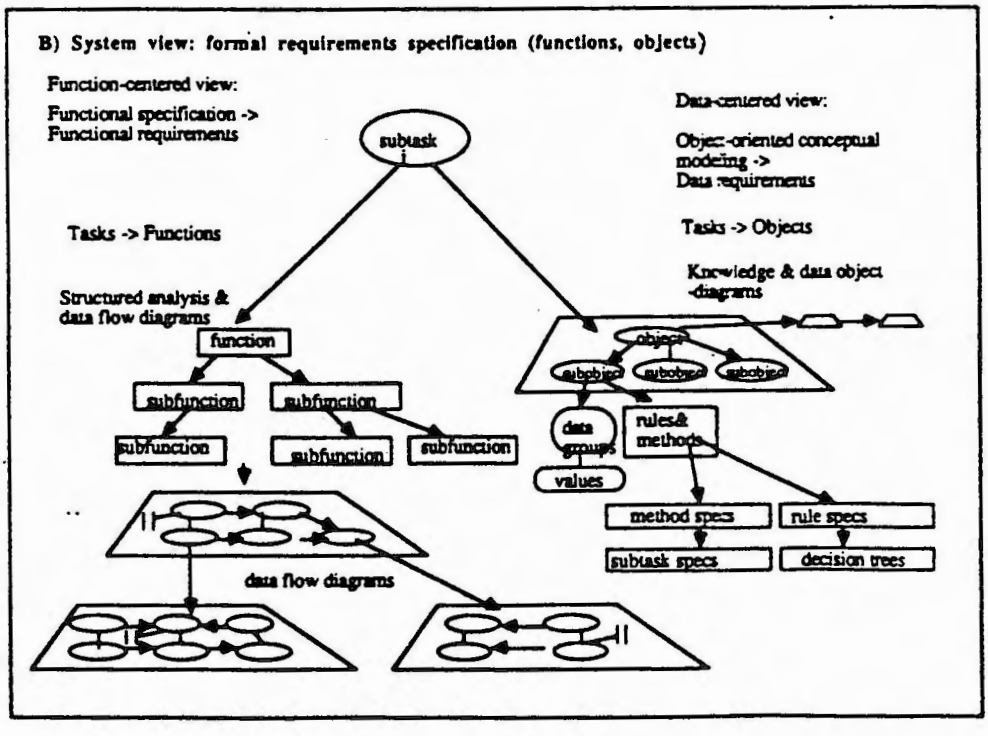
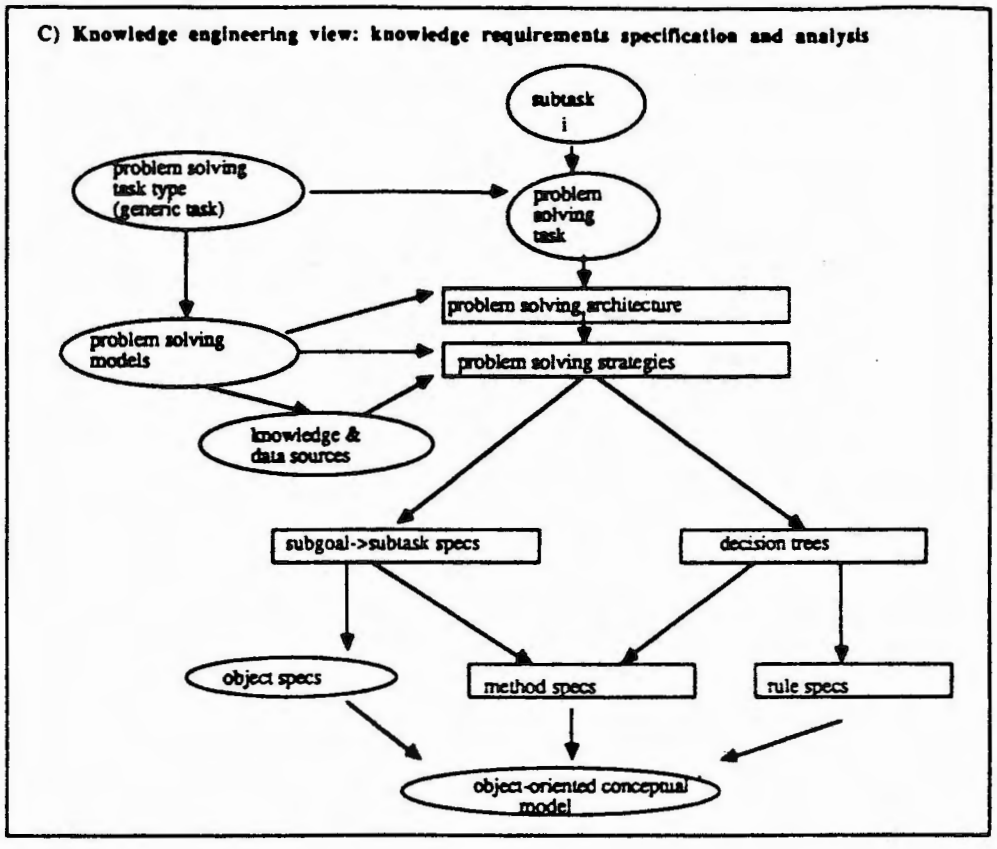
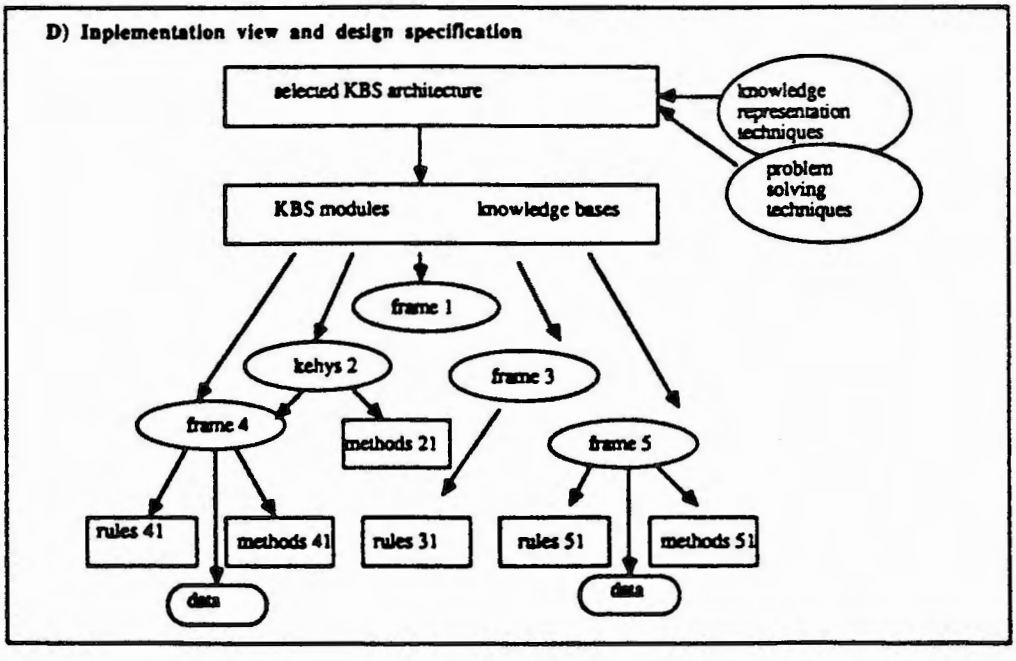


Figure 1. The different levels and viewpoints of the requirements specification of a KBS (a,b,c,d).


C) Knowledge engineering view: knowledge requirements specification and analysis
subtask i
problem solving task type (generic task)
problem solving task
problem solving architecture
problem solving models
problem solving strategies
knowledge & data sources
subgoal->subtask specs
decision trees
object specs
method specs
rule specs
object-oriented conceptual model


D) Inplementation view and design specification
selected KBS architecture
knowledge representation techniques
problem solving techniques
KBS modules
knowledge bases
frame 1
kehys 2
frame 3
frame 4
methods 21
frame 5
rules 41
methods 41
rules 31
rules 51
methods 51
data
data

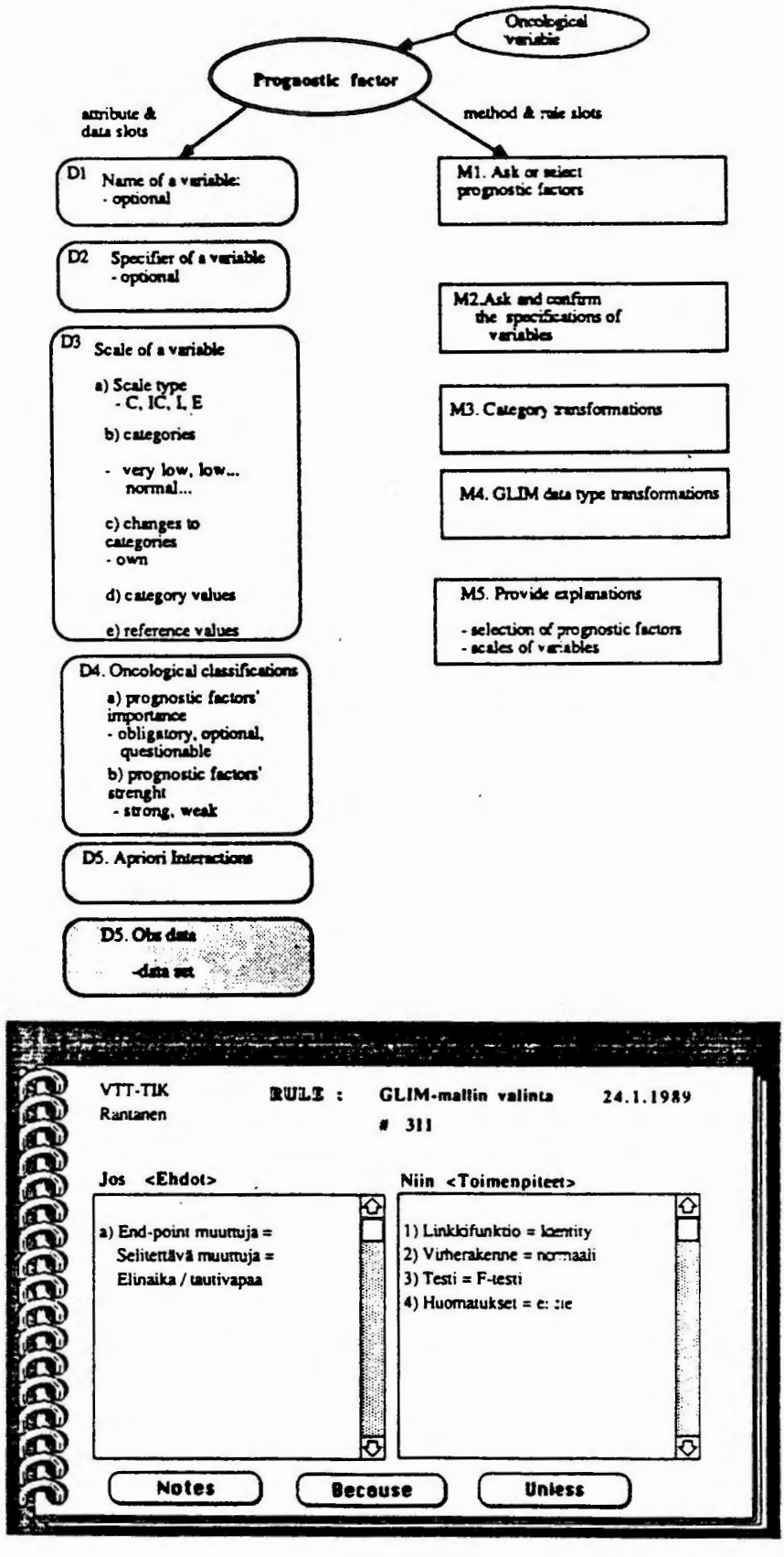


Figure 2. An informal specification of an object of the intelligent user interface for a statistical package (above). This kind of informal diagrams (initially outlined as wall charts) were starting points for more detailed hypermedia-based specifications, such as the decision rule description below.

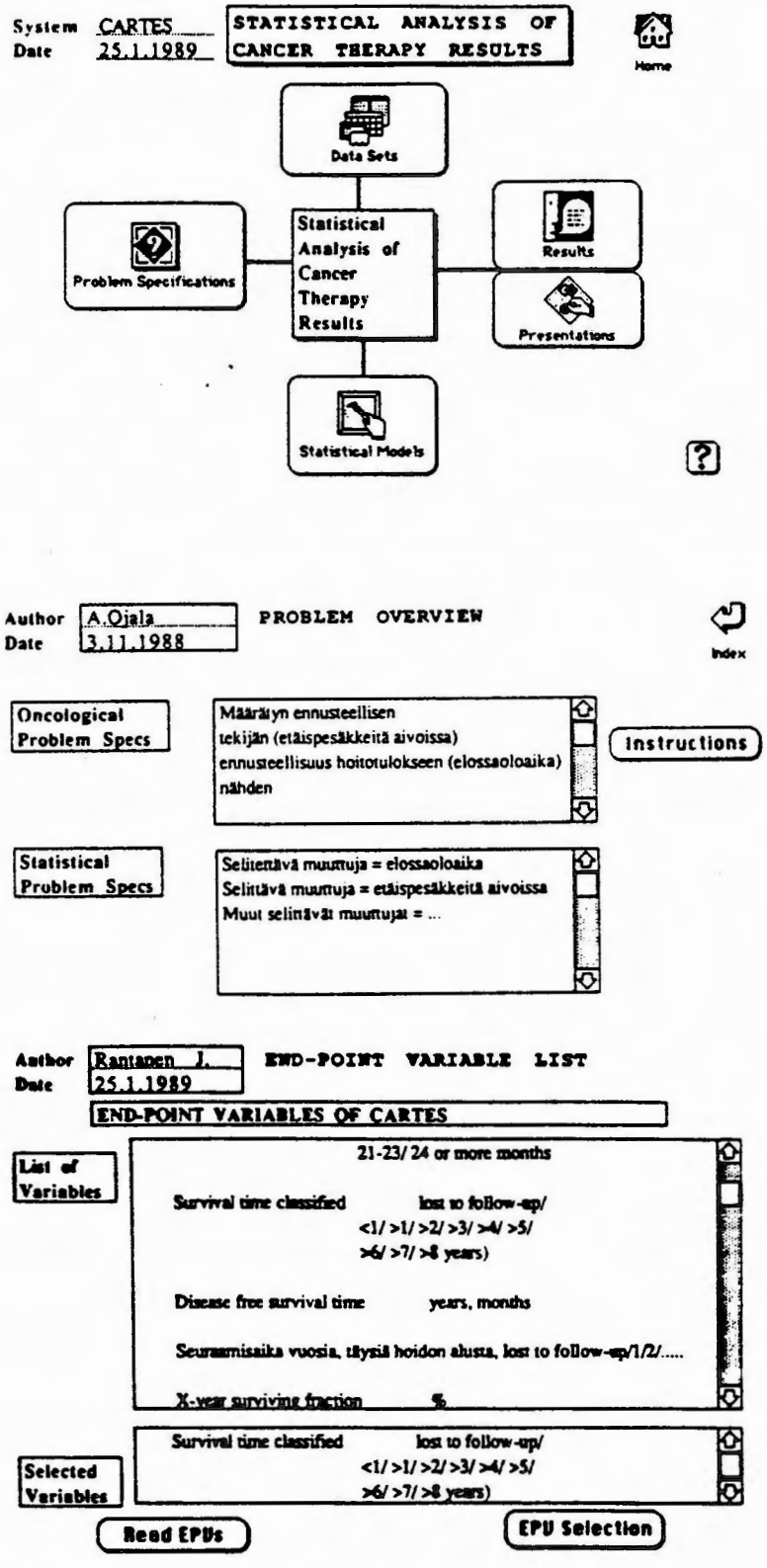


Figure 3. Some displays from the HyperCard-based implemenation of the intelligent user interface of the CARTES KBS.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Sortal analysis with SORTAL, a software assistant for knowledge acquisition

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

SORTAL is a software assistant for performing meaning-triangle–based sortal analysis. This paper describes its architecture and possible implementations. Conceptual analysis and conceptual modelling are central components of the informant-and-analyst–based, natural language–mediated knowledge acquisition process, but focusing on concepts is not enough. The "aboutness" of the language used in the interview forces the analyst to recognize distinctions between words, concepts, referents, and cogniting agents. Creating frame-like representations for agent-centered meaning triangles, as well as updating ontologies, keeping track of multiple domains of discourse, and the creation of knowledge bases for use in other systems are tasks that can be assisted by a software tool such as SORTAL. We sketch the requirements for such an assistant, give examples of its operation, and address implementation issues.

1 INTRODUCTION

Knowledge acquisition is difficult if for no other reason than that the analyst has to keep track of a vast bulk of textual and conceptual material. Software tools are essential. In this paper we describe a software system, SORTAL, a knowledge acquisition assistant for sortal analysis, that helps the analyst in her work. The development of knowledge-based software is merely a special case of a more general issue of software engineering: how can we use CASE (computer-assisted software engineering) tools to make our work easier in systems analysis, requirements specifications, and data modelling?

SORTAL is one of three theoretical or implementable software systems that we are developing. Together, they address the problems of text handling, conceptual analysis, natural language understanding, and the extraction of knowledge from text. The systems are these:

- LUKES (Language Understanding and Knowledge Extraction System) is a theoretical, "gedanken" system for generating expert systems. LUKES is based on a three-level architecture of 'cogniting' agents, having verbal, conceptual, and subconceptual levels. It incorporates a theory, based on concept cluster attachment, of natural language understanding and knowledge acquisition from text. We describe this system and its theoretical foundations more fully in [Regoczei and Hirst 1989b].
- LOGOS ("the word, the expressed concept") is an implementable version of LUKES. In its present form, LOGOS has a two-level architecture: verbal and conceptual; *i.e.*, it omits the subconceptual level of LUKES. It is knowledge-based, and is bootstrapped by SORTAL (which we discuss next) (see figure 1). Its input is natural language text and descriptions of conceptual structures. As output, it produces knowledge bases consisting of text with concept cluster attachments.
- SORTAL, the main topic of this paper, is a KA analyst's tool that assists in the use of the KA methodology that we have described in earlier papers [Regoczei and Plantinga 1987, Regoczei and Hirst 1988, 1989a, 1989b]. In particular, SORTAL uses the meaning triangle distinctions of verbal, conceptual, external-referent, and agent domains of existence discussed in [Regoczei and Hirst 1988]. It helps the analyst to build ontologies and inventories for domains of discourse from informant-analyst

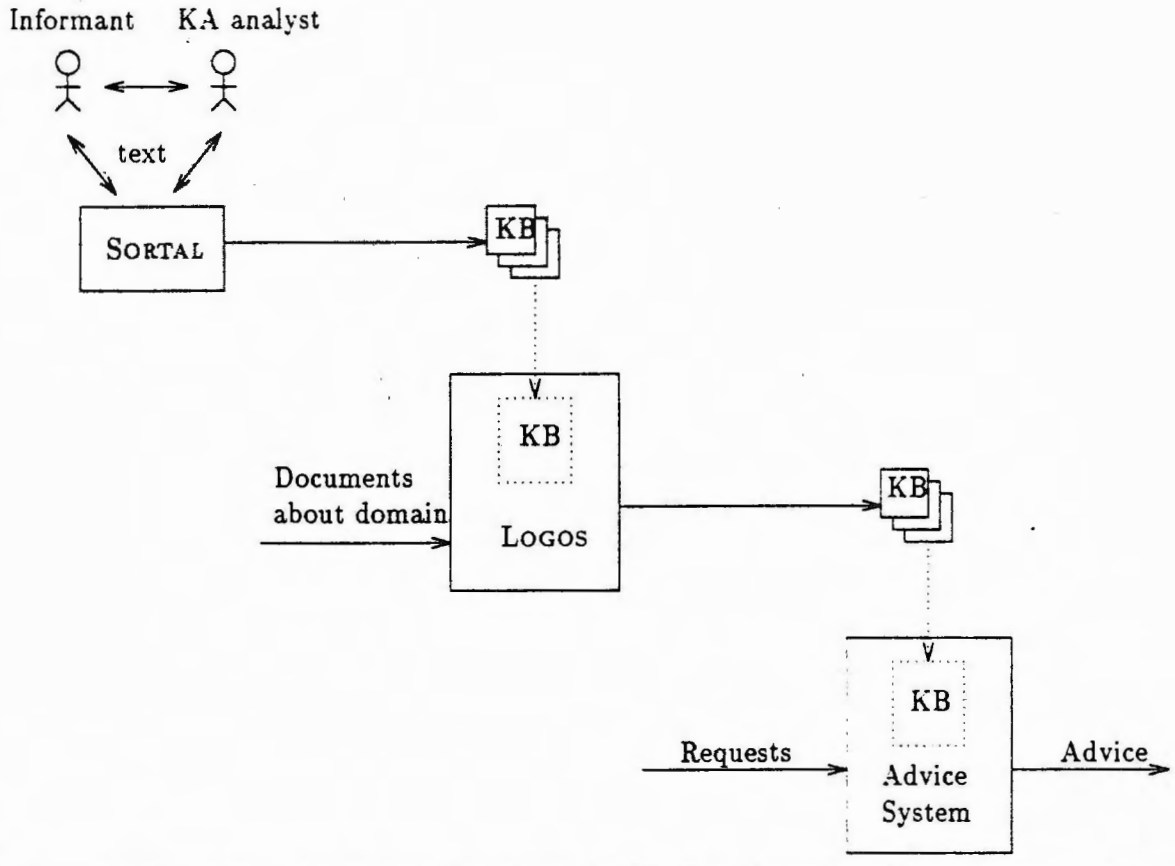


Figure 1: Bootstrapping LOGOS with SORTAL to build a knowledge-based system. (From Regoczei and Hirst [1989b].)

interviews, other textual material, and direct observation of the object domain. It incorporates version control for domain of discourse creation, and other features described below.

The problem we are addressing is to build conceptual models on the basis of natural language input to the analyst, making use of the analyst's existing knowledge base, and, having done this, to take the conceptual structures so obtained and recast them in the framework of one or more of the following:

- Knowledge base construction for expert systems;
- Requirements specifications for software development;
- Data modelling for conceptual schema design;
- Systems analysis.

We see both SORTAL and LOGOS as kinds of CASE tools that integrate some of the best current techniques with the new theoretical foundations that we are offering. We observe that there is a good deal of software development being done right now: some of it is done well and some of it is done badly. Our view is that it should be possible to form a descriptive theory of what is being done in the field, examine it and refine it, and formulate a systematic methodology that one could recommend. A methodology for KA, of which we picture sortal analysis *(see next section)* as one component, is best embodied in software tools. SORTAL is an example of the approach that we are taking.

2 SORTAL ANALYSIS

Our approach to knowledge acquisition was described in the earlier papers cited above. In these papers, we argued for the importance of creating a *domain of discourse*, and record-

ing the *ontology* and *inventory*, in the context of an informant-and-analyst–based, natural language–mediated interaction. We advocated the *meaning triangle* (figure 2), as a way of drawing distinctions between entities with different ontological status. We advanced the mechanism of *concept cluster attachment* to operationalize both knowledge extraction from text and natural language understanding. In connection with the meaning triangle and concept cluster attachment, we discussed *sortal analysis*.

Sortal analysis is a form of classification. From input of text, general signs, or other observational material, it separates and clearly labels words, concepts, referents, and agents. Sortal analysis can be regarded as a generalization of conceptual analysis. In the case of conceptual analysis one is inquiring about the concepts 'behind' the text. But we have to go beyond this. We need to distinguish between a language in the abstract sense, such as English, and 'language-in-use', such as the utterance of an English sentence to communicate with another person or to perform a task. Language-in-use is *about* something. In the case of sortal analysis, we are inquiring about what *sort* of things are being talked about, as well as recognizing text itself as a sortal category, namely verbal. Sortal analysis recognizes the central importance of the 'aboutness' of language-in-use.

Sortal analysis consists of generating and constructing concepts, concept clusters, and conceptual structures, forming mental models of the conceptual kind, categorizing, and contextualizing, while recognizing the fundamental distinctions between words, concepts, and entities that are components of an object domain in the external world.

For example, the sense of a word can be defined as an ordered pair, whose first component is lexical, *i.e.*, the word, and whose second component is a concept cluster centered upon some concept. For example:

⟨"piano",
⟪⟪[PIANO] | [MUSIC], [SONATA], . . .⟫⟫ ⟩

The central concept, before the vertical bar, is the *denotation* and the peripheral concepts of the cluster are the *connotations*.

A second example: *Sortally composite entities* are frames. In perceiving a table, we can decompose the activity into perceiving, perceiving that something is there, and attaching the concept [TABLE] to the perception. If we verbalize the perception, then we have a triple:

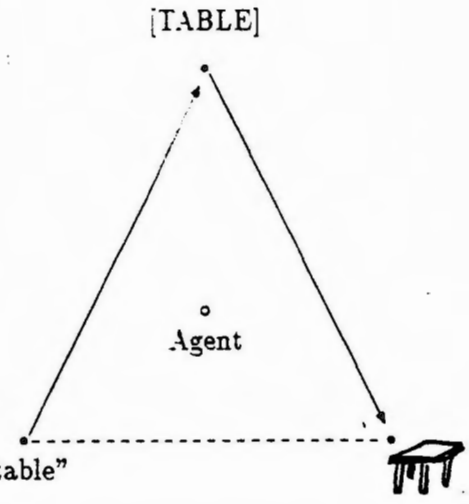


Figure 3: Meaning triangle for a sortally composite entity.

⟨  , [TABLE], "table"⟩

We can picture this triple as a frame. Such frames can be diagrammed on a meaning triangle with the contents of the individual slots diagrammed on the referent, concept, and word nodes, respectively, as in figure 3.

Sortal analysis is necessarily *perspectivist* [Bullock *et al* 1988, p. 641–2], *constructivist*, and *agent-oriented*. One cannot do sortal analysis on a sentence alone without further information and knowledge about people and the world. We will illustrate this point with an example below.

In sortal analysis, our first step is to take the agents seriously and feature them prominently, because the distinction between concept-like and referent-like entities is to be drawn relative to an agent. This categorization of entities is essential. We can note the two-level architecture in the case of the NLU approach: The English text is at the verbal level, and the con-

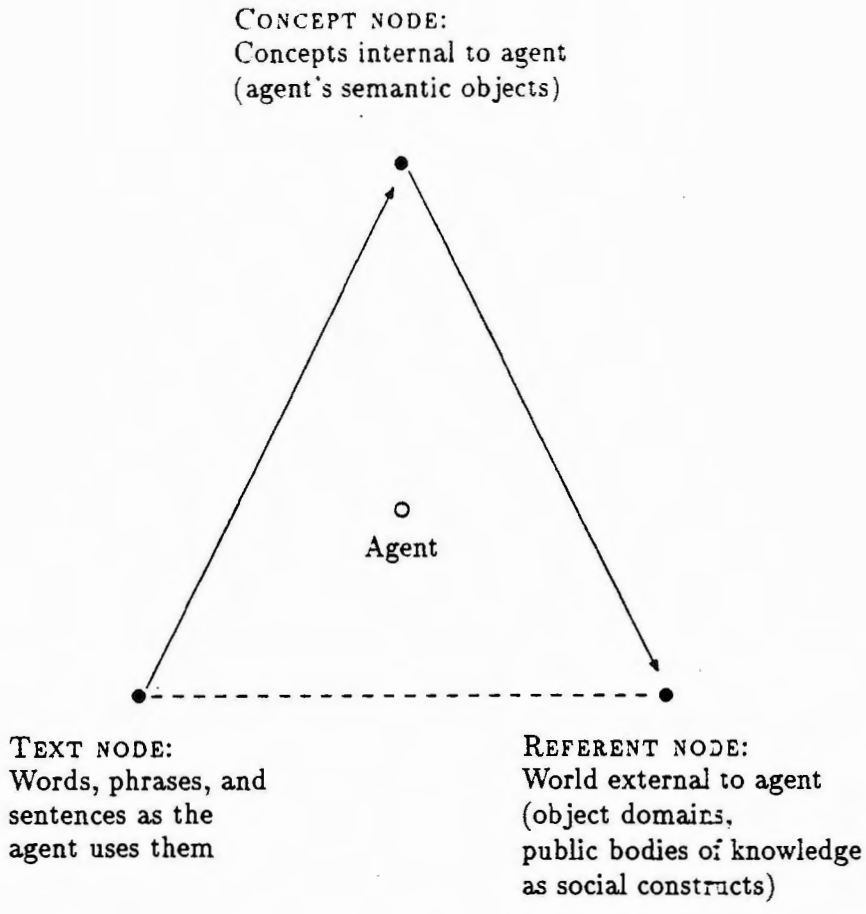


Figure 2: An agent-centered meaning triangle.

ceptual dependency representation is at the conceptual level. But this does not exhaust the ontological categories. In knowledge acquisition, we have to take cognizance of agents and the world they live in. In particular, we have to use surrogates (see section 7) to construct models of the object domains.

3 SOFTWARE SUPPORT FOR SORTAL ANALYSIS

Even for relatively simple KA problems, the volume of material to be handled far exceeds the capabilities of manual systems. Certainly, automated techniques are more cost-effective. SORTAL is a software assistant for the task of sortal analysis. It supports the recording of concept cluster attachments and the diagramming of the ontology and inventory on frame-like versions of the meaning triangle. Figure 4 lists SORTAL's main functions.

SORTAL's being an assistant means that the person being assisted is also a part of the system. This has to be noted, because concept formation is something presently done by people, in this case the analyst, and not by computers (current research in AI notwithstanding). It is the analyst who decides what is an agent, how many triangles are needed, and how surrogates will be created for referent entities in the object domain. SORTAL records, organizes, and prompts, enabling the analyst to keep track of large quantities of textual and conceptual material.

4 HOW SORTAL HELPS

To put it very simply, SORTAL sets up frames, and prompts the user to fill the slots.

What makes SORTAL significant for our purposes is the distinction drawn in categorizing entities into verbal, conceptual, agent-like, and referent-like. Furthermore, it distinguishes public concepts from other entities on the referent node, and supports the construction of sortally composite entities.

- Helps analyst draw meaning triangles.
- Supports
  the construction of agent-centered meaning triangles in frame-like notation.
- Supports domain of discourse creation.
- Records ontology.
- Records coreference links.
- Provides version control.
- Effects categorization into verbal, conceptual, referent-like, and agent-like entities.
- Supports the construction of sortally composite entities, *e.g.*, word–concept cluster pairs, or named concepts, or conceptualized referents.
- Keeps track of hybrid entities on the referent node, for possible future refinements of the analysis.

Figure 4: Main functions of SORTAL.

When confronted by a piece of text, the analyst asks questions such as: who is uttering this text, what are this person's mental models, what is the text referring to, and is there in fact a referent? That is, she makes a distinction between understanding text, understanding people, and understanding the world [Regoczei and Hirst 1989b]. She makes the decision as to how many and what kind of agent-centered meaning triangles to start with, and whether or not to include a triangle centered upon herself as an agent. Next, she decides which text fragments to deal with, how compositional the concept cluster attachment should be, and whether she should follow a literal interpretation or a more figurative one. In setting up multiple domains of discourse, she separates out the various conceptual domain components on the concept node of the meaning triangle. She decides whether or not to include *platonic domains* either on the concept node or on the referent node.

SORTAL helps to populate the referent domain with surrogates. Because the object domain on the referent node is structured by projecting concepts onto the node, a judicious policy has to be adopted to trade off:

1. The inclusion of sufficient surrogates not only to cover the referents of the concepts but also to provide a rich-enough background for breakdowns and unexpected deviations [Winograd and Flores 1986];
2. Not including so many surrogates that the inventory becomes unmanageable.

In deciding what to include on the concept node and what to place on the referent node, the guiding consideration is the trade-off between models of mental models and models of 'the truth'. This is the great thematic distinction between 'objective' and 'subjective', and it takes a certain amount of humility on the part of the analyst to realize that what appears to her as 'God's Own Truth' is little more than a verbalizable version of a conceptual model, which, in turn, models her own mental models, some of which may be at the subconceptual level.

In the case of an agent-centered meaning triangle, there are at least two agents: the agent in the center of the triangle, and the analyst constructing the triangle. We have to distinguish between the agent structuring the object domain and the analyst structuring the object domain. This is a non-trivial distinction. The analyst and the agent may see the world very differently, *i.e.*, from different perspectives. SORTAL can keep track of such distinctions. Thus not only does it keep track of multiple domains of discourse for a single agent, but it has to keep track of different views of the 'same' domain, whether the object domain or the domain of discourse.

5 AN EXAMPLE

To illustrate sortal analysis and SORTAL, we will now present a situation that we will use as a running example, interwoven with our theoretical discussion. We will consider a sentence similar to one originally analyzed by Schank [1973, p. 228] to demonstrate his conceptual dependency representation. We want to highlight how different a KA analysis of text is from that taken in conventional research in natural language understanding (NLU). The text is:

(S1) John sliced the salami with a knife.

Schank produces a semantic representation of this in his CD notation, employing his set of

primitive actions; see figure 5.[1] Being able to produce such diagrams and then use them to answer questions is said to constitute language understanding.

Our approach necessarily starts not with sentences but with utterances—that is, the particular use of a sentence by an agent in a particular situation in the world. Our analysis takes into account a consideration of the background to the utterance: certain cultural or anthropological observations and certain pragmatic considerations.

Firstly, in North America, salami usually comes shrink-wrapped or sliced at the delicatessen. Very few people slice salami. They hardly even slice bread any more. They do know, however, that salami comes sliced. The slicing is done not with a knife but with a machine with a large rotary blade that can be set for different thicknesses of slices. Moreover, many people would, if pushed, realize that they are confused by the notion of salami. They would have difficulty telling salami from Polish sausage or from jagdwurst. For example, is bologna a type of salami? What about Genoa salami? The relevance is that bologna is relatively easy to slice with a knife, but soppressata is not. Schank didn't have to worry about these issues because there he had no real salami slicing. His main concern was computational linguistics of a 'pure' kind—agentless, timeless, contextless, ethereal—whereas for us dialogue is there to acquire knowledge possessed by an agent and used in an actual situation for a particular purpose. (Yes, we are using a pretend world too, in order to make our example small enough to fit in this paper, but we are nonetheless aiming at some measure of realistic and dramatic verisimilitude.)

The second observation concerns discourse rules. The sentence seems 'unnatural' as a fragment of actual dialogue. It is contrived; one could almost say that it was written in a kind of artificial natural language. In the case of a more casual dialogue, one would set up a context such as one in which John and some other people were preparing some cold cuts, in order to make a point such as this:

- John did the slicing.
- He was slicing salami, not bread.
- He used a knife, because he was skilled enough to be able to cut thin, even slices for the sandwiches.

This is a lot of new information to squeeze into a single sentence. Hence it stretches the bounds of the rules of co-operative discourse [Grice 1975]. The point is not that a sentence such as *John sliced the salami with a knife* would never be uttered; obviously it can be uttered, and is a perfectly acceptable, reasonable, well-formed English sentence. But without setting the context, establishing the agents, talking about the various referents, situations, and domains of discourse, one can do neither knowledge acquisition nor sortal analysis. So, according to the rules of discourse, the speaker must supply enough information, but not too much, and must make sure that the utterance is interpretable. In other words, the sentence has to be contextualized.

As we said, this example is originally from research in NLU. To make it a KA problem, we must put it in context. Let us assume that we are automating the sandwich production facility of a large catering company. We talk to the supervisor, asking about the source of some cold cuts on a plate, and he casually says:

> (S1) "John sliced the salami with a knife."

Now we have to 'extract' the knowledge from this sentence. (For the operationalizability of this metaphor, see Regoczei and Hirst 1989b].)

6 WORKING THE EXAMPLE BY HAND

6.1 A first version

What would this example look like if, instead of using Schank's approach, we tried conceptual analysis techniques? We would get a much richer analysis, and results that are much more suitable for knowledge base construction, although conceptual analysis still falls short of

[1] Schank's example was actually *I sliced the meat with a knife*. But his analysis of *John sliced the salami with a knife* would be exactly the same as his actual example but for the tokens *John* and *salami* replacing *I* and *meat*. In contrast, as may be inferred from the discussion below, the two sentences would give rise to very different analyses in our approach.

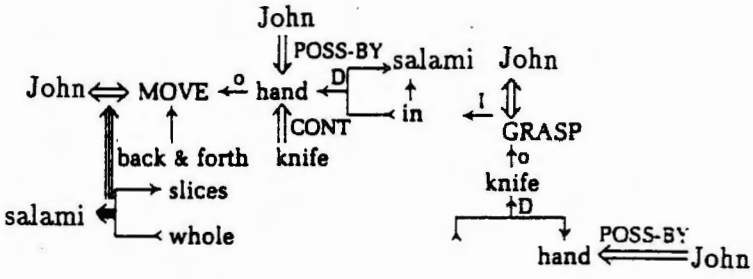


Figure 5: Schank's representation of *John sliced the salami with a knife.*

sortal analysis, as it addresses the ontology, but not the inventory, of the particular case or situation.

Let us try to work this example 'by hand', using conceptual analysis to explicate the knowledge 'behind' the text [Regoczei and Hirst 1989a] and use Sowa's conceptual graphs to record the results of the analysis [Sowa 1984]. Later on, we will introduce sortal analysis techniques. A simpler analysis takes sentence S1 as the text, generates the concepts closely corresponding to the lexemes, and arrives at a graph such as that in figure 6. This graph—let us call it Version 1—says, basically, that the activity of slicing has an agent, John, an object, the salami, and an instrument with which it is being carried out, the knife.

The ontology for our microdomain at this stage consists of four concepts, three conceptual connectives, and one statement of fact regarding John's activities. At this stage, we know nothing further about [KNIFE] and [SLICE], so for the moment they are to be considered semantic primitives relative to the microdomain we are building. Knowledge about knives or the activity of slicing may come from the informant or may be supplied from the private knowledge base of the analyst. For proper version control, we should be able to keep track of where this additional input is coming from. We will return to this very important point below.

6.2 Recording the resulting knowledge

Now let us consider the difficulties that the analyst faces in the task of making a permanent record of the interim findings. Until recently it had to be done by paper and pencil, or equivalent. Alternatively, one could use a word processing package and the frame-like notation for meaning triangles suggested by Regoczei and Hirst [1988]. With a tool such as HyperCard, the analyst could actually draw triangles on a set of linked hypertext cards (figure 7).

But this is not much of an advance, as it still has some serious shortcomings, such as these:

- There is no connection between the entities in the text, concept, and referent fields. In particular, there are no coreference links between these entities.
- There is no support for forming sortally composite entities such as the ones that we need to clarify the example of Eddington's two tables [Regoczei and Hirst 1989b]:

  ⟨#table, [TABLE-COMMONSENSE]⟩
  ⟨#table,
  [TABLE-NUCLEAR-PHYSICS]⟩

  The projection of concepts onto the referent node is necessary to structure the object domain there, and to label the surrogates. Alternatively, we could attach descriptive labels on the surrogates in a metalanguage which, of course, would look exactly like English. This causes a confusion—responsible for much philosophical head-scratching in the past—between the population of the text node and the explanatory labels attached to the surrogates.
- There is no facility for treating the populations of the text, concept, and referent nodes as different abstract datatypes with different permissible operations. In particular, the items on the nodes are not

[JOHN]←(AGNT)←[SLICE]→(OBJ)→[SALAMI]

↓

(INSTRUMENT)→[KNIFE]

Figure 6: Conceptual graph for sentence S1.

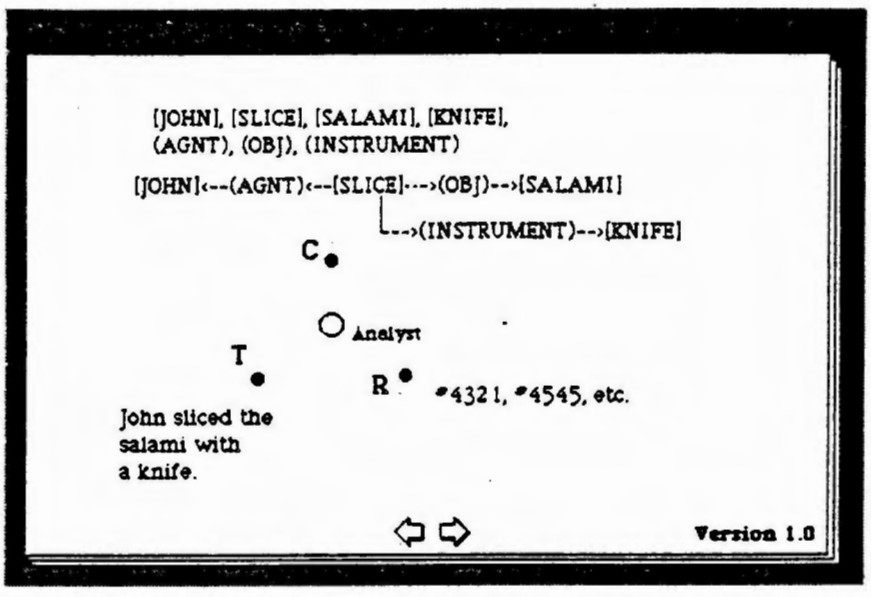


Figure 7: Version 1 of the analysis, hand-diagrammed in HyperCard.

available for further processing, such as, for example, generating knowledge bases from the information already available.

- There is no support for intuitively-needed human operations such as concept cluster attachment [Regoczei and Hirst 1989b]. Attaching concept clusters to words and referents is very much like free association or being reminded of one thing by another. As such, it is a subjective, agent-oriented activity not unlike forming coreference links or sortally composite entities, or the labelling of referent surrogates (see section 7 below).

This brief list makes it clear that a software tool supporting sortal analysis would have to provide substantial advantages beyond manual techniques, even when the manual techniques are supported by word-processing, database, and text handling capabilities.

6.3 Continuing to work the example by hand

Resuming where we left off above, we can look at the conceptual graph that expresses the factual statement of John's activities (figure 6) and perhaps notice some peculiarities having to do with the naming of concepts. We notice, for example, that the concept [SLICE] refers to the activity of slicing, rather than the piece of salami that is the final product of the activity. Ideally, the names of concepts should be intuitively obvious both to analyst and informant, and anyone else later reading the documentation of the interviews. We could decide to use the name [SLICING] instead, though Sowa does not use the gerundial form in his graphs. In any case, using the gerund consistently for an activity is unnecessary, because difficulty arises only when a word, such as "slice", is both a verb and a noun designating the product of the action. It might be easier to explain the concept with comments, rather than to agonize over the exact choice of the name. In casual terms, we

would like to elaborate and explicate—in effect, talk about the concept—including explaining that possible labels such as [SLICE-ACTIVITY] and [SLICING] are alternative names for the same concept that 'defines' a type of activity out there in the 'real world'.

When there is no software support, the considerations we just described, although occurring in the thoughts of analyst and informant, do not get recorded. A text window, associatable with each concept, is a necessity for recording these explanatory comments. The text so generated may, at a later stage in the interview, be incorporated in one of the later versions.

Comparing our graph above with Schank's (figure 5), we notice that Schank's is more complicated. Its complexity is introduced by:

- Elaborating on the activity of slicing, in particular, slicing with the knife.
- The salami being turned from an integral, whole entity into a collection of smaller pieces. Schank, however, stops short of describing each piece as having a particular shape (*i.e.,* being delimited by parallel planes) and a specified orientation relative to the salami as a whole.

We can incorporate similar considerations into our analysis by drawing canonical graphs (à la Sowa) for the concepts [SLICE-ACTIVITY] and [SLICE-OBJECT]. We can also consider adding further information through *type hierarchies* such as:

SLICE-ACTIVITY
< CUTTING-ACTION
< ACTION

or considering adding concepts with referent fields (not the same as surrogates!) such as:

[PERSON: John]

and:

[CUTTING-ACTION: John's-slicing]

We may broach the issue of whether the distinction between action and activity needs to be specified at this stage, the intuition being that slicing is more like a prolonged activity than a single act or action. However, there is no need to take a rigid position on what may be included in the microdomain. Without a predetermined directory of concepts and a taxonomy, the analyst has to make decisions as the microdomain is being constructed on what to include and in what form to include it. But it would be useful to have a software tool to keep track of these decisions for future use, possibly to scroll back to an earlier version to construct a different account of the knowledge being accumulated. With manual techniques, as noted above, the thinking is done, but is too cumbersome to record. Even if recorded, it is too difficult to reuse.

Focusing our attention now on the use of semantic primitives, we note that the primitives that we use are not absolute, but rather are relative to the microdomain being constructed. Since we are striving for a minimal covering ontology of sentence S1. at times we make the decision to treat a concept as a one-node primitive, and not expand it in terms of further canonical graphs. Schank takes a more absolutist position on what the 'true' primitives are. But Schank's primitives may have more to do with his own psychological "thematic preferences" [Halton 1973] than with any platonic, true, or real structure of knowledge. For example, the action of slicing is expanded in terms of primitive actions and hence is not a primitive. But, we may ask, why is *knife* a primitive in Schank's version? Why are actions to be explicated, but objects not? Is this because Schank is an active person? There is nothing simple about knives, nor about objects in general. We can think of some typical cases of knives as being things that are sharp-edged, and have a handle and possibly a point. We have strong subconceptual notions about knives and different kinds of knives. Many of these notions are conceptualizable. But some of the concepts used in building private conceptual models of knives do not have names. In talking about them, people usually resort to expressions such as "you-know" and "thingamajig".

To keep the analysis within bounds, it is often wise to make an explicit decision on how much detail to include. This is to be construed as a design decision, and not in Schankian terms of having reduced the analysis to some collection of atomic elements that cannot be decomposed any further. For many purposes, there is no point in elaborating any further than to note, perhaps in the form of text added to Version 2,

that:

> "The knife has a handle to grasp and a sharp edge to cut with."

Thus, at the Version 2 stage we can introduce concepts such as [HANDLE], [GRASP], [EDGE], [SHARP], [CUT], etc. We introduce them first by increasing the text, and then seeking to construct the covering ontology. Other techniques would be to do concept cluster attachment [Regoczei and Hirst 1989b] and then elaborate on the concepts introduced. As we are developing larger and larger minimal covering ontologies for the text available, we exercise version control. We have version control; Schank does not. However, enforcing version control with manual techniques requires the kind of self-discipline that usually cannot be attained in a production environment.

To conclude: At this stage we have two versions of the analysis. Version 1 was shown earlier in figure 7. Version 2 appears in figure 8. The name that appears as the agent's name in the triangle is that of Stephen Regoczei, one of the authors of this paper, because it was he who did the analysis. Had it been done by someone else, or had it been done by Regoczei at a different time, the results might have turned out somewhat differently. (As in data modelling for databases, there are no unique answers, only pragmatic design decisions.)

7 SURROGATES FOR REFERENTS

Probably the greatest difference between what we are doing and what the NLU approach does relates to surrogates. "John" is the name of a person, and that person is an entity with the concept [PERSON] attached to it. However, the entity to which the lexical object "John" and the conceptual object [PERSON] are attached—if such an entity exists at all—is made of flesh and blood and cannot be placed into silicon chips. So let us agree to appoint a unique character string, such as #35467829, which will 'stand in' for this flesh-and-blood entity, acting as a surrogate. Thus our meaning triangle will appear as follows (in frame notation):

> **Agent:** Stephen Regoczei
> **Word node:** "John"
> **Concept node:** [PERSON], [JOHN]
> **Referent node:** =35467829

Now we can form some sortally composite entities:

> ⟨#35467829, [PERSON], "John"⟩

This says that the entity whose surrogate is #35467829 is being considered a person by Stephen Regoczei, and the name of this entity is "John". Furthermore, the concept [JOHN] bears a relation to the concept [PERSON]. The exact nature of this relationship is, however very difficult to state, because there are several different ways of thinking about it. For example, we could say that John is an *instance* of a person. In Sowa's graph notation [Sowa 1984], this would be expressed as

> [PERSON: John]

Or we could say

> [JOHN]←(IS-A)—[PERSON]

and leave the meaning of IS-A appropriately vague.

We can continue looking at the relationship between the concept and the referent nodes by noting that the concept [JOHN], when projected onto the referent node, is mapped onto #35467829. We give 'meaning' and 'significance' to the uninterpreted string #3546789 by attaching the concept [JOHN] to it.

We can extend our meaning triangle by establishing coreference links and by listing newly constructed sortally composite entities. These entities are not diagrammed on a vertex; they can be pictured as being located on the 'sides' of the triangle. Adding the relation

> Coreference ("John", [JOHN], #35467829)

would indicate that these character strings all refer (in a sense!) to the 'same thing', but they are not the same thing! Words, concepts, and referents are different. Much misunderstanding in knowledge acquisition work, and in interpersonal communication, is caused by the faulty use of coreference links.

Sortal composites are n-tuples, such as word–concept pairs or surrogate–concept–name triples. Thus we can expand our meaning triangle frame by adding, for example, a slot for composite entities:

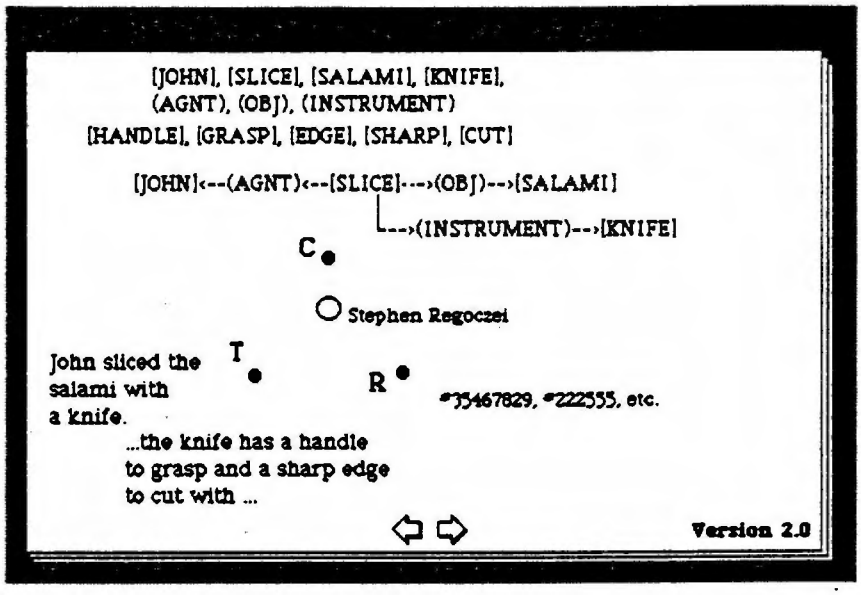


Figure 8: Version 2 of the analysis.

Composite entities:

Word-concept: ⟨"John", [JOHN]⟩

Referent-word: ⟨#35467829, "John"⟩

Referent-concept:
⟨#35467829, [PERSON]⟩

Word-concept-referent:
⟨"John", [JOHN], #35467829⟩

Coreference links are created by the analyst as an annotation on the triangle, but the sortally composite entities are construed as being 'out there'.

In the example *John sliced the salami with a knife*, is there a referent for "John" in the NLU approach? Probably not. The sentence was a piece of fiction; its author was talking in a 'pretend' sort of way (*cf* our own disclaimer above). For a pretend world, there are no surrogates. At best, substitute surrogates are constructible as possible or potential entities by projecting the pretend concepts onto the referent node.

Which entities require surrogates depends on the focus of attention. In the sentence about John's salami-slicing activity, there are few clues about focus and emphasis. Suppose the point to be conveyed were the following:

> John knife-sliced the salami, rather than machine-slicing it.

or:

> John sliced the salami—with a knife, no less!

The point is that this time around the mental models of the agent are different because the focus on a knife-like action emphasizes the type of action being performed but not the instrument. Which particular knife was used is no longer of significance. So in constructing surrogates for the entities in the object domain, it may or may not be necessary to construct a surrogate such as #222555, specified by the sortally composite entity:

⟨#222555,
"the knife that John actually used"⟩

Similarly, in contrast to the conventional NLU analysis, changing *a knife* to *the knife* commits us to very different surrogate structures on the referent node. We would then have to identify the particular knife. Often we are faced with the unknown structure of the world, and either accept the vagueness or have to hypothesize surrogates as projections of concepts generated by an agent. In the next section we discuss techniques for coping with vagueness.

8 DISCOURSE ANALYSIS IN SORTAL ANALYSIS

The analyst must pay attention not only to what is said but also to what is not said. She

must decide what seems to be missing and try to fill in the blanks. There are many different kinds of lacunae, each requiring a different treatment. Three kinds are particularly important:

- Recovering missing verb modifiers. The modifiers of a verb are its argument positions: the agent, patient, instrument, time, reason, place, and so on. (Not all verbs allow all kinds of modifiers.) For example, the utterance "The salami was sliced" leaves open the questions of how the salami was sliced and by whom.
- Making presuppositions and tacit assumptions explicit. Utterances frequently presuppose much more than they directly assert. For example, "The sandwich hand sliced the salami" presupposes that there is a person employed as a sandwich hand whose role probably requires further investigation.
- Guessing at information withheld by the informant. Compare these two exchanges:

  "Is the system cheap and high quality?"
  "Yes."

  "Is the system cheap and high quality?"
  "Yes, it's cheap."

  It is well known that informants will withhold information for good reasons of their own. Ferreting out this information is crucial for knowledge acquisition, and the success of the project is largely dependent upon the analyst's skill.

We plan that SORTAL will have a facility to include notes and additional textual and conceptual material by the analyst to supplement the text uttered by the informant. This is important not only in cases when attention is paid to situations such as those we described, but also in the cases of ellipsis, hard-to-resolve anaphora, metonymy, metaphor, and other uses of figurative language. SORTAL will facilitate the generation of explicit augmentations both at the textual and the conceptual levels, tag them as such, and provide an 'audit trail' of attributions.

Prompting for the filling of lacunae is easier than is commonly thought. For example, in impersonal phrasing, an inanimate object is substituted for a cogniting agent, such as in this sentence [Regoczei and Hirst 1989a]:

The car smashed through the barrier.

Driven by whom? Was it, perhaps, driverless? If John was driving the car, there is a difference between saying this sentence and saying that:

John smashed through the barrier.

The end result, in terms of the referent or state of affairs in the world, may be the same, but there is a difference at the conceptual level and the issue of volition and culpability may come up. So a distinction has to be noted. In logical terms, the identity of "John" and "the car" may become an issue. The possibility of such apparent anomalies may lead one to consider that the sentence contains figurative language—in particular, metonymy. This kind of possibility may be detected with relatively low-level, syntactic and lexical techniques. The system need not 'know' too much about the concepts and the referents behind "John" and "the car" to tag them for further investigation.

In the current version of SORTAL, some prompting facility will be built in for textual augmentation. These facilities are necessarily limited at present, because the creation of enhanced facilities would depend on two crucial prerequisites. First, research in natural language understanding, especially in discourse analysis and pragmatics, has to be further advanced. Second, the provision of the augmentations is knowledge-based and discourse dependent. We see the generation of such supplementary material as akin to concept cluster attachment. As we argued in an earlier paper [Regoczei and Hirst 1989b], concept cluster attachment is a knowledge-based activity that requires considerable specific knowledge on the part of a cogniting agent. To enhance SORTAL's ability to prompt, we need, in effect, to add small expert-system modules to it that contain expertise regarding the kind of discourse analysis techniques and the kind of missing verb modifiers that are appropriate to prompt for in specific circumstances. We describe these difficulties not only to specify the current limitations of SORTAL, but also to point out the important research directions for the future that are precon-

ditions for further enhancement of the present version.

9 A TYPICAL SESSION WITH SORTAL

We will now briefly outline what a typical session with SORTAL would look like, following this example. We want to emphasize that the sequence of interaction, *i.e.*, the user–SORTAL dialogue, is not prewired. Each user can set up the kind of prompting sequences that best suit her cognitive style. There would be a library of these interaction templates available: verbose; brief; superficial analysis; deep analysis; concentrating on the verbal, conceptual or referent node; multiple agents; multiple triangles with joins and unification; multiple domains of discourse for the same agent; emphasizing coreference links, sortally composite entities, or the platonic domain; and so on.

In our case, the *dramatis personae* include the analyst, the supervisor, John, and an impersonal super-agent probably representing the company or the policies and procedures of the company from whose view the 'correct way' of doing things gets formulated.

We have to decide who the 'cogniting' agents are. John may play no greater role than slicing salami, in which case he is not a cogniting agent. On the other hand, if his understanding of whether Genoa salami or summer sausage is to be knife-sliced or machine-sliced matters, then his mental models are to be recorded on an appropriate meaning triangle centered upon him.

As the analysis proceeds, a consensus emerges. For recording this consensus, a consensus triangle, with supervisor and analyst on the agent node is to be created. The population of the concept node in this consensus triangle may be transferred to the policies-and-procedures triangle, as judged or deemed appropriate by a joint decision of supervisor and analyst.

A typical session with SORTAL may start as follows:

> *SORTAL:* Establishing meaning triangles. What criteria would you like to use?
>
> *Analyst:* Agent.
>
> *SORTAL:* List agents.
>
> *Analyst:* Me, Supervisor, Consensus, Corporate
>
> *SORTAL:* Which triangles are the input triangles?

Here SORTAL displays the frames for the triangle to allow the analyst to indicate which triangles are the input triangles. The analyst uses a mouse to indicate that her own triangle and the triangle of the supervisor are input, the consensus triangle is an intermediate workbench or breadboarding device and the corporate triangle is for the 'objective knowledge' that the knowledge base is intended to capture and present. The analyst indicates that the consensus triangle has joint agents consisting of herself and the supervisor.

SORTAL continues:

> *SORTAL:* Which triangle would you like to work on?
>
> *Analyst:* [Indicates supervisor's triangle]
>
> *SORTAL:* Text?

The analyst enters the text in a separate window on the screen and uses a mouse to carve the text into fragments. As she touches various fragments of the green text, yellow versions of the text fragments, indicating the corresponding concepts. are transferred to the concept node. Likewise. she can create blue surrogates, which are then transferred to the referent node. There are two styles of display for SORTAL's main working screen. The first is intended to be more convenient for use by the analyst, while the second, a more triangle-like display, is for reviewing the contents of the fields with the informant; see figure 9.

The supervisor-centered meaning triangle created by the analyst reflects her mental models of the supervisor's mental models. This is an important point. The supervisor may not agree with the analyst's interpretation. If he does, the agreed-upon material should be transferred to the consensus triangle. But usually a discussion takes place about the areas of disagreement. This dialogue should be carefully recorded for future analysis. Before this, however, a version-control procedure is activated.

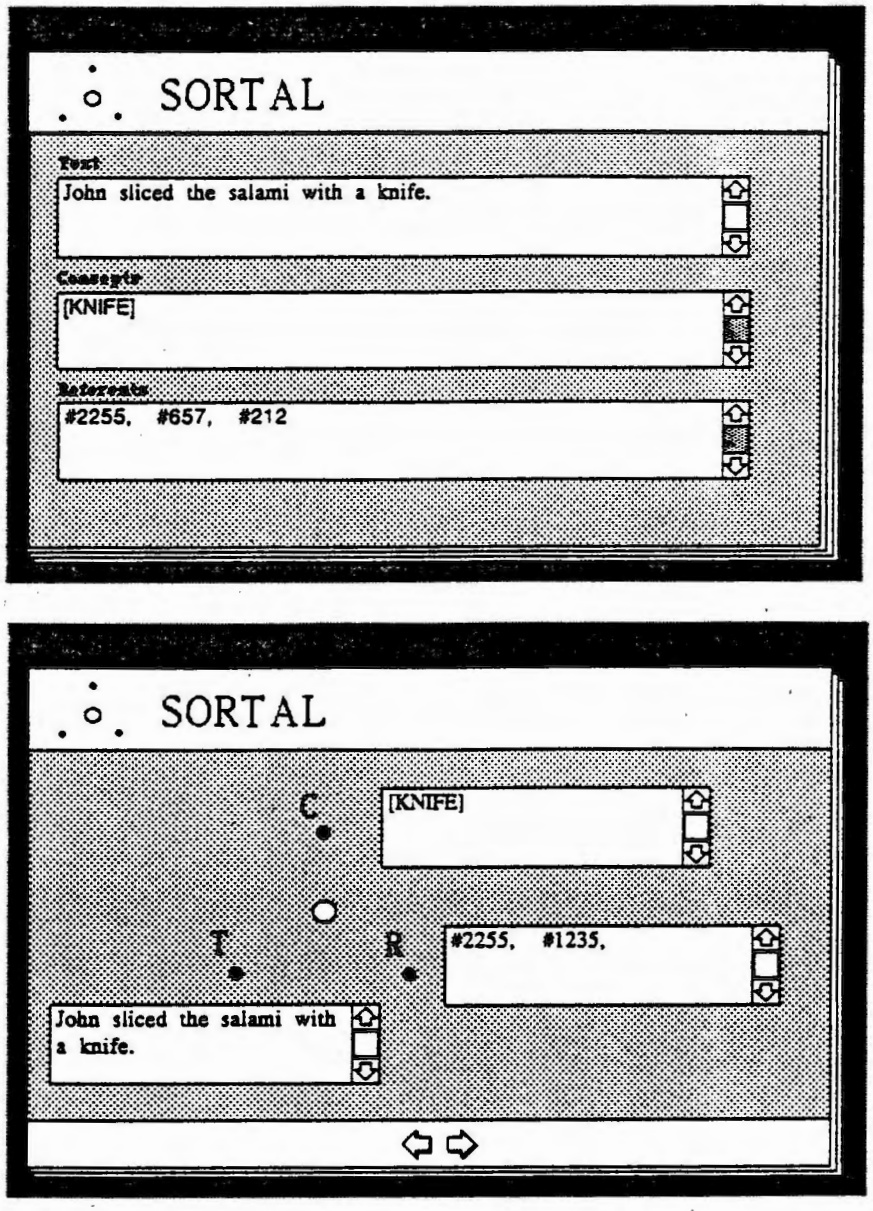


Figure 9: Two styles of display for SORTAL's main working screen.

The meaning triangles are transferred to a version library and are labelled as Version 1.

The analyst inquires further about the details of the situation to which the sentence refers. Eventually, the following comment may arise:

> *Supervisor:* No, John wasn't doing it properly. In fact, now that I think about it, it probably wasn't John after all. It was somebody else—I don't know who. [Hence no surrogate is needed.] The point is that hard salamis have to be sliced on the machine, because otherwise we can't control thickness. With softer things like bologna, they can use the knife instead if the machine would have to be cleaned.

The analyst and supervisor are now working towards the consensus triangle, which may contain a rule such as this:

> If salami is hard, then use the machine.
>
> If salami is soft, then preferably use the machine; as an exception, one may use a knife if the machine needs cleaning first or would need cleaning afterwards and this would result in an unacceptable time delay.

After further consultation, SORTAL may be used to transfer this text to the word node of the corporate triangle, together with the conceptual translation of it either in conceptual graph notation, frame notation, logical notation, or Telos (see section 10 below). The 'objective' knowledge base would be generated from the corporate triangle by taking the contents of the word node and attaching concept clusters from the concept node [Regoczei and Hirst 1989b].

10 IMPLEMENTATION OF SORTAL

A common error in designing software is to distort the architecture by a premature consideration of what can be easily implemented. Thus often the compromise system is the first and only design that ever gets produced. Here and in our other papers we try to distinguish clearly between:

- What *must* be built as the essential core of the system (the "must-have" list).
- What one would also *like* to build (the "nice-to-have" list).
- What *can* be built given the state of the art and resource limitations.

At present, there are not enough utilities available to implement the SORTAL architecture in a cost-effective way. Basically, we need concept processing rather than word processing. The main capability needed is being able to go back and forth between diagrams and text easily. Hence, ideally we would need a hypertext-, database-, or frame-based knowledge representation facility. Probably the closest approximation would be expert systems shells such as Goldworks or KEE, with an extension whereby one can generate concepts and referent surrogates and also interface with graphics diagramming. Ideally, we would have a database with screen handling and freeform icons and diagramming, with a facility to attach text and concepts to parts of the diagram. Until such utilities become available, we can emulate the implementation of the SORTAL architecture—even using simple word-processing packages, relying upon a user-imposed discipline.

At present, we are investigating the feasibility of several different alternatives for embodying a substantial kernel of SORTAL functions. The alternatives are, in decreasing order of desirability (as we see it at the moment):

1. Hypertext, with a strong graphics capability so that diagrams can be fragmented and concept clusters attached to the fragments. This is a capability that could be also used to fragment sentences and associate concept clusters with the fragments. Icons generated with the graphics facility could be used as surrogates. (There are advantages to using such non-textual surrogates.)
2. A state-of-the-art knowledge representation language such as the Telos system being developed at the University of Toronto [Koubarakis *et al* 1989a, Koubarakis *et al* 1989b].
3. An expert-system shell with the appropriate knowledge-structuring capabilities.
4. An already-existing KA tool that could be adapted for our purposes.
5. An already-existing CASE tool that could be adapted for our purposes.

6. A relational database with a fourth-generation language interface with screen-management capabilities for setting up forms.
7. A text file, under an editor, using certain human-enforced, disciplined notational conventions.

At present, we are investigating options 1, 2, 3, and 7 to establish a range for what is feasible and cost-effective. We are prototyping SORTAL in HyperCard, and planning to do a production implementation in Smalltalk. For the analysis work, we have been using conceptual graphs to record our findings on the concepts node. Conceptual graphs are well suited for knowledge representation at the analysis stage because they are concept-oriented and their surface form is not entirely dissimilar to natural language as actually used by analyst and informant. But for the purposes of processing the population on the concept node, we would want to transfer to a language like Telos.

We can summarize gradually more enhanced implementations of the SORTAL architecture by outlining five levels of practicing sortal analysis:

1. Essentially manual: The analyst draws pictures of meaning triangles on the screen using a tool such as HyperCard. There is no processing of what appears.
2. HyperCard prototyping: Separate fields for the text, concept, and referent nodes, and support for coreference links and sortally composite entities (as seen in figure 7). The contents of fields are available for some limited processing.
3. SORTAL implemented with Smalltalk processing behind a HyperCard user interface: Implementation of populations on the nodes as abstract data types.
4. SORTAL with Telos: To give fuller manipulation capabilities in the concepts field. A conceptual-graphs-to-Telos interpreter is required.
5. Interaction between the natural-language level and the conceptual level: Three-way communication between the text node, conceptual graphs, and Telos, to output knowledge bases in the form of text–concept pairs.

The features of the implementations will be tested on a large body of material gathered by the first author over years of systems analysis, data modelling, and knowledge acquisition work.

11 KNOWLEDGE ACQUISITION MODULES FOR KBMSs

Knowledge-base management systems (KBMSs) are systems that assist in the creation, management, and use of knowledge bases [Brodie and Mylopoulos 1988, Schmidt and Thanos 1989]. SORTAL is a tool for building knowledge bases that may be passed on to a KBMS for further management. But there is another, more unified way of looking at things. SORTAL can be considered a part of the KBMS: its knowledge acquisition module.

In considering SORTAL as a KA module for a KBMS, we need to think about the different types of KA modules possible. In particular, we need to distinguish between software *assistants* for KA and *automated* KA modules. The specifications for these two types are very different. The main purpose of a software assistant is to enhance the capabilities and improve the performance of the people using it. The software is not expected to produce conceptual structures or knowledge bases all by itself. The people using it are very much part of the system, and it is this system as a whole that produces the final product. SORTAL is such a software assistant. Its effectiveness is partly to be judged on the basis of how well it enables people to do their jobs. Automated modules, on the other hand, are expected to work reliably, independently of people's input. No human intervention should be required. LOGOS [Regoczei and Hirst 1989b] is an example of such an automated module for the acquisition of knowledge from text.

11.1 Processing the results

The output of SORTAL is triangles with a population on each node. Each population can be considered as being of an abstract data type, to be manipulated according to appropriate admissible operations. The population on the text node is to be handled by standard text-processing or natural language–processing packages. The population on the referent node is to

be structured by organizing it into object domains, and structuring these domains by projecting the concepts from the concept node onto the referent node.

The population on the concepts node is like a knowledge base. One form of output, as mentioned before, consists of conceptual structures together with natural language equivalents attached. Another way of handling the population of the concept node is to transfer it into a language such as Telos, thereby making it usable with a KBMS.

To summarize: The population of the text node is text, to be handled as such. The transition from the text node to the concepts node is concept cluster attachment. The population of the concepts node, translated into Telos, is handled by a KBMS. The population of the referent node is structured by projecting from the concepts node, either by forming sortally composite entities or through coreference links. Explanations can be attached to referents by forming referent–text pairs. Furthermore, the comments of the analyst can be attached as text windows to referent surrogates, as they can be to text fragments, conceptual structures, and sortally composite entities.

12 CONCLUSION

In this paper, we have discussed ideas concerning sortal analysis, the use of the meaning triangle, and concept cluster attachment. However, the main purpose of this paper is to outline how to put these ideas into action, by embodying them in software tools. We described the requirements for such a tool, SORTAL, described desirable features of the architecture, reviewed the implementation problems, and suggested a feasible emulation of the most vital components of the architecture.

To repeat the main point: The acquisition of knowledge from text relies not only on the understanding of text, but also the understanding of people and the understanding of the object domain. Concept cluster attachment, the hypothesized mechanism for attaching knowledge to text, brings about the generation of a great deal of textual and conceptual material. The recording and organizing of this material requires software assistance.

An additional complication is introduced by taking seriously the 'aboutness' of language-in-use, and hence recording not only concepts but also an adequate inventory of surrogates for entities that populate the referent domain. To make this task feasible, better software tools are to be developed for analysis and conceptual modelling. We believe our research to be contributing to this continuing development.

Acknowledgements

For discussions on these topics we are grateful to Tom Addis, Bob Carter, Marc Linster, John Mylopoulos, Doug Skuce. and Steve Twine. Our research is supported by grants from the Natural Sciences and Engineering Research Council of Canada, the Trent University Committee on Research, and the Information Technology Research Centre (an Ontario Government Centre of Excellence) through its KBMS project (John Mylopoulos, principal investigator).

References

[Brodie and Mylopoulos 1988] Brodie, Michael L. and Mylopoulos. John (editors) (1986). *On knowledge base management systems.* Springer-Verlag.

[Bullock *et al* 1988] Bullock, Alan; Stallybrass, Oliver; and Trombley, Stephen (editors) (1988). *The Fontana dictionary of modern thought* (second edition). London: Fontana Press.

[Grice 1975] Grice, H. Paul (1975). "Logic and conversation." In: Cole, Peter and Morgan, Jerry L. (editors). *Syntax and semantics 3: Speech acts.* New York: Academic Press, 41–58.

[Halton 1973] Halton. Gerald James (1973). *Thematic origins of scientific thought: Kepler to Einstein.* Cambridge, MA: Harvard University Press.

[Koubarakis *et al* 1989a] Koubarakis, Manolis; Mylopoulos, John; Stanley, Martin; and Jarke, Matthias (1989). "Telos: A knowledge representation language for requirements modelling." Technical report KRR-89-1, Department of Computer Science, University of Toronto, January 1989.

[Koubarakis *et al* 1989b] Koubarakis, Manolis; Mylopoulos, John; Stanley, Martin; and Borgida,

Alex (1989). "Telos: Features and formalization." Technical report KRR-89-4, Department of Computer Science, University of Toronto, February 1989.

[Regoczei and Hirst 1988] Regoczei, Stephen and Hirst, Graeme (1988). "The meaning triangle as a tool for the acquisition of abstract, conceptual knowledge". *Proceedings, Third Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, November 1988. To appear in *International journal of man-machine studies*, 1989.

[Regoczei and Hirst 1989a] Regoczei, Stephen and Hirst, Graeme (1989). "Knowledge acquisition as knowledge explication by conceptual analysis." In: Sowa, John F; Foo, Norman; and Rao, Anand (editors). *Conceptual graphs for knowledge systems.* [To appear.] Also published as technical report CSRI-205, Computer Systems Research Institute, University of Toronto, January 1988.

[Regoczei and Hirst 1989b] Regoczei, Stephen and Hirst, Graeme (1989). "On 'extracting knowledge from text': Modelling the architecture of language users." *Proceedings, Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems*, Paris, July 1989, 196–211. Also published as technical report CSRI-225, Computer Systems Research Institute, University of Toronto, January 1989.

[Regoczei and Plantinga 1987] 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. Reprinted in: Boose, John H. and Gaines, Brian R. (editors). *Knowledge acquisition tools for expert systems* (The knowledge-based systems series 2). London: Academic Press, 1988, 293–308.

[Schank 1973] Schank. Roger Carl (1973). "Identification of conceptualizations underlying natural language." In: Schank, Roger Carl and Colby, Kenneth Mark (editors). *Computer models of thought and language.* San Francisco: W.H. Freeman, 187–247.

[Schmidt and Thanos 1989] Schmidt, J.W. and Thanos, C. (editors) (1989). *Foundations of knowledge base management.* Springer-Verlag [to appear].

[Sowa 1984] Sowa, John F (1984). *Conceptual structures: Information processing in mind and machine.* Reading, MA: Addison-Wesley.

[Winograd and Flores 1986] Winograd, Terry and Flores, Fernando (1986). *Understanding computers and cognition: A new foundation for design.* Norwood, NJ: Ablex Publishing.

PM: A METRICS DRIVEN PLAN COMPILER

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Robert G. Reynolds, Jonathan I. Maletic, and Stephen E. Porvin

Wayne State University
Department of Computer Science
Detroit, Michigan 48202

Draft

1. INTRODUCTION

Herbert Simon suggests that one of the major bottlenecks to the generation of automatic programming systems is the acquisition of programming knowledge for use in such systems [Simon 86]. For example, a number of researchers have demonstrated the importance of planning knowledge in the programming process ([Soloway 88], [Letovsky 86], [Rist 86], [Kant 81]). Several automatic programming systems employ plan-like knowledge ([Rich 88], [Barstow 79], [Johnson 87]). The library of plans associated with a given knowledge-based system is the most important factor influencing that system's performance. However, acquisition of the planning knowledge that goes into such a library can involve an extensive amount of handcrafting on the part of the engineer. Such handcrafting is time consuming and impacts the nature of plan-based systems in three fundamental ways: First, it can narrow the scope of domain applications. Second, it can limit the number of applications within a given domain. Third, it can limit the number of target languages supported by the planning system.

The Programmers Apprentice system offers a good illustration of the amount of effort involved in adding a plan to a knowledge base [Rich 88]. There, the knowledge engineer provides the system with a *cliche* or pseudocode extension of a given target language. Figure 1 contains an example of a report generation cliche for the Ada target language taken from [Rich 88]. The terms enclosed in brackets, {...}, correspond to stubs in a pseudocode program that need to be filled in with information provided by the user or from another cliche. This cliche is then transformed using the system's knowledge-based editor, KBEmacs, into an associated graph structure that can be reasoned about using a plan calculus.

In terms of this example there are two basic obstacles to the efficient input of such plans. First, a substantial proportion of completed code in the target language may have to be entered. Steps must then be taken to insure the syntactic and semantic correctness of that portion of the cliche's specification. Since this code is not completed code, such tests will be difficult to perform. Second, the variables comprising the role names must be expressed in terms that correspond to cliches already present in the knowledge base, which potentially requires global knowledge of the knowledge base's current contents. While neither of these obstacles should pose problems to the system at the prototype stage, they can make the development of a mature system, one that deals with multiple domains, multiple applications within a domain, and multiple languages, a difficult chore.

The goal of the Partial Metrics project (PM) described in this paper is to investigate the metrics-driven acquisition of planning knowledge. The approach taken here can be thought of as that of reverse compilation. Abstract plans are generated from a more detailed source program in a specified language. This process involves the integration of traditional compiler techniques with certain AI ideas. The principle phases of the process are as follows:

1). Generation of Code Aggregates
In this phase, the source code is broken up into a collection of code segments called *aggregates*. These aggregates can be thought of as the tokens generated by the lexical analysis phase of a compiler. The aggregation procedure used here is based upon code complexity metrics [Reynolds 86] and language complexity metrics [Reynolds 88a]. The *granularity*, or size of the aggregates, is determined by the metric constraints that are applied and influences the kind of generalizations that are made.

2). Aggregate Hierarchy Generation
In this phase, properties of each aggregate are described in a frame notation and a hierarchical collection of aggregates is produced based upon these properties. The placement of an aggregate in the structure depends upon its level of nesting inside other aggregates as well as a metrics based assessment of its structure. The result is an ordered tree in which the aggregates that are most likely to be abstracted are in position to be visited first during the generalization or chunking process to follow. This phase is analogous to the generation of a parse tree in a traditional compiler.

3). Chunk Generation
In this phase, certain aggregates are selected for chunking based upon their metric properties. Section 2 gives a brief overview of the role that chunking has played as a problem solving tool in AI. The result of generalizing an aggregate is to insert in its place a set of symbolic constraints. These constraints may then be propagated to other aggregates to insure consistency. This phase is akin to the semantic analysis and code generation phases of the compilation process.

4). Chunk Storage and Activation at "Refine Time"
In this final phase the chunked hierarchy of frames is stored in the knowledge base for use during the code generation process by a hierarchical planner.

The following sections discuss each phase at length. In order to demonstrate how this approach can contribute to existing plan-based systems, the target code generated by the Programmers Apprentice described earlier is used as input. This example is chunked at several levels of granularity and the results compared with the original cliche.

2. THE ROLE OF CHUNKING IN PM

2.1 The Chunking Process

The need for humans to associate specific input with more general or abstract patterns was motivated by the work of Miller, a psychologist, on the size of a human's short term memory [Miller 56]. His results suggest a limit on the size of an average person's short term memory of seven plus or minus two items. This limitation can be overcome when each of the objects in short term memory is allowed to represent an encoding of a collection of more detailed objects. For example, few people would be able to remember the string of bits 010110101100. However, if the string is linearly decomposed into three 4-bit subsequences, (0101, 1010, and 1100), then each 4-bit sequence can be encoded in its hexadecimal equivalent. The resultant set of objects, 5 A C, is easy to remember. These new numbers can serve as input to some analytical process that can produce

a solution in terms of the chunked representation. Here, for instance, the process may just be the modulo 16 addition of these 3 objects which yields B. This result can then be decoded into the original, more detailed, representation if necessary.

The above example suggests that chunking consists of several distinct phases. The first phase is that of decomposing the input into aggregates of low-level objects. These aggregates should be relatively independent in order to simplify the encoding process. It was Simon's observation [Simon 69] that many complex systems can be decomposed into a collection of hierarchically structured subsystems, such that the intra-system interactions are relatively independent of inter-system ones. The decomposition of the binary sequence can be expressed hierarchically as :

```
          010110101100
            /   |   \
        0101   1010   1100
```

If there is no dependence between the inter- and intra- subsystems, then the system is said to be *completely decomposable*, as is the case for the example above. *Nearly decomposable systems* exhibit a few interactions, but still permit efficient encoding to take place [Simon 69].

The second phase of the chunking process is the encoding or abstraction phase. If the aggregates are independent then the encoding process is particularly straightforward since the abstraction mechanism can be applied independently to each. The presence of some interaction between components may cause the propagation of symbolic constraints produced by the chunking of one aggregate to others. In a nearly decomposable system this propagation should be relatively local in nature.

The third phase involves the use of this collection of chunked aggregates in some problem solving process. It is assumed here that the collection of chunked aggregates can be described in a hierarchical manner and that the problem solver is a hierarchical planner.

The chunking process described above has been employed in a variety of AI applications. For example, chunking has been used as a vehicle for a hierarchical planning system for chess [Campbell 88]. In addition, chunking has been used as a basis for learning about problem solving procedures [Laird 86], and as part of a general model of human practice [Rosenbloom 83].

2.2 How the Chunking Process is Integrated into the Structure of PM

In figure 2 the basic phases of the plan compilation process are related to the generic chunking paradigm. Notice the number of phases in the PM compiler that support the generalization of the hierarchy of aggregates (HAG). Special attention is paid to the construction of the aggregate structure in our system in order to reduce the effort expended in the encoding or generalization phase.

The aggregation procedure is driven by a set of metric parameters which are input with the target code to be processed. These metrics, which are discussed in more detail in section 3, were selected based upon their ability to successfully describe patterns in the structure of code segments added to pseudocode programs during the code refinement process.

The relative independence of the set of aggregates produced is then assessed in metric terms. If the aggregates do not exhibit enough structural independence then the aggregation process can be repeated by adjusting the values of the metric parameters to produce a more appropriate grain size for the aggregates. The problem of finding a combination of metric values that produce aggregates with acceptable levels of independence is an example of the "granularity problem" described by Mostow and Roy [Roy 88].

The advantages inherent in dealing with the granularity problem prior to the generalization process are two-fold. First, it reduces the effort involved in generalization since fewer interactions between aggregates need to be handled. Second, it results in the generation of plans that are efficient to use. Newell and Tambe have shown that some chunks are more expensive to employ in the problem solving process than others [Tambe 88]. One of the causes of this expense is the presence of a large number of condition elements in the chunk. By using metrics to force the relative independence of aggregates, it is anticipated that the number of such conditions, and therefore the cost of using the aggregates, is reduced as a result.

2.3 What is being Chunked in PM?

Input to the PM system is target code from a specific programming language. The current prototype supports the Ada, Pascal, C, OPS5, and Prolog languages. The target code example used here is the Ada code generated by the Programmers Apprentice system as a result of instantiating the roles in the simple report cliche given earlier. Figure 3 presents the code listing.

In addition to the language constraint, the prototype requires that the target code have the following properties:

1) The program is produced using structured programming techniques.
2) The target code should have been verified or at least well tested.

Since the plan generated from the code is used as the basis for many other designs, this constraint prevents the "broadcasting of errors" to other applications that use the plan. The sections that follow demonstrate how a target program that satisfies the above constraints is processed by the PM system.

3. THE METRICS-BASED GENERATION OF A HIERARCHY OF AGGREGATES

The principle AI paradigm employed in this section is that of search. Simon has suggested that the implementation of target code via the stepwise refinement of pseudocode can be naturally expressed in terms of state space search [Simon 86]. The state space to be searched here is the set of pseudocode programs in a target language. Given an initial abstract description of the program to be produced, the programmer searches for a sequence of refinement steps that will produce suitable target code. A tree representing the possible paths from the initial pseudocode description to a final code module is given in figure 4. This tree is called a *refinement tree* by Barstow [Barstow 79]. Each node in the tree corresponds to a pseudocode program, and an arc from a node A to a node B represents the fact that B was produced by a refinement of A.

For the implementation of any given program, each refinement in the programming sequence can be recorded to give the actual path taken through the tree by the programmer. In figure 4 the specific path taken by the programmer is highlighted. Studies of numerous refinement sequences in the software engineering literature suggest a tendency of programmers to place constraints on the path taken through the tree in terms of the code complexity added at each step. ([Reynolds 87a, Reynolds 88b]).

A set of metrics was developed to characterize these constraints on the search process. These metrics are designed to assess the structural complexity of the code aggregates added during the refinement step. Several

basic assumptions were made in the construction of these measures:

1). Each refinement step can be viewed as an aggregation of a number of constituent code generation decisions. A portion of each decision deals with the expression of the results in terms of an internalized model of the target languages syntax. Therefore, the syntactic structure of the target language's grammar affects the aggregation decisions made. This perspective has been suggested by others such as MacLennan [MacLennan 84]. The metrics selected to represent the impact of the languages grammar on the refinement process are called *refinement metrics* [Reynolds 88b].

2). The nature of the code aggregate added at each refinement step is also influenced by limitations on the programmers short term memory. It is assumed that these limitations can be estimated by counting the lexical symbols added to the code at each refinement step. The metrics developed to assess this aspect of the code aggregate are called *partial metrics* [Reynolds 87b].

3). Both partial and refinement metrics should also codify information used by a programmer in selecting a path of refinement through the tree that satisfies certain limitations on the code complexity added at each step. In doing so, the metrics must be capable of predicting the complexity of the completed code that would be achieved if the programmer were to follow the current path to completion. If the prediction exceeds some acceptable limit the search systematically shifts to some other candidate path. It is important to insure that the prediction consistently underestimates the possible value. If this is not the case, then the fruitful path may be ignored [Winston 84]. Both refinement and partial metrics have been designed to allow the prediction of properties along an entire path. In addition these estimates are designed to consistently underpredict the complexity of the completed code associated with the current path.

The goal of the aggregation process here is to traverse the refinement tree in reverse by systematically removing an aggregate of code from the target program and replacing it with a stub. The aggregate that is removed satisfies the set of constraints provided as input. This process continues to construct a path through the refinement tree from the bottom up. The result of the process is the production of a hierarchically nested sequence of code aggregates where each aggregate satisfies the given metric constraints. Note that this path does not have to correspond to that used to construct the program in the first place. This is demonstrated in figure 4 where the program is shown to be produced via one path through the tree, and the decomposition occurs in reverse following another.

Here a simple breadth-first search from bottom-to-top is used to generate the path. The search is guided by an evaluation function composed of two metrics, one each from the set of partial and refinement metrics. These metrics were selected because of their observed ability to describe the structure of the code aggregates added to the code at each refinement step.

3.1 The Metrics Used for Aggregation

Since the goal here is to generate a reverse path through the refinement tree, it is desirable that the aggregates produced during the traversal should be subject to similar constraints as those in the original path. If this is the case, the aggregate structure produced is more likely to be understandable to a human observer. In order to understand how each metric is able to characterize a particular aspect of the code aggregate added, the basic effects of a refinement decision on the code must be described.

The external effects of a refinement decision can be manifest in two basic ways:

1. A change in the syntactic structure of the pseudocode program.
2. A change in the implementation subgoals associated with the pseudocode program.

In order to measure the effect of a refinement decision it was necessary to produce measures that quantitatively assess the changes of each effect. *Partial metrics* were developed as a means to describe changes in a pseudocode program's structural complexity [Reynolds 84]. A partial metric is computed relative to a specific model of a pseudocode or partial program. In terms of this model, each program consists of a *projected* part and a *prescribed* part. The prescribed part corresponds to reserved words and symbols in the target language.

The projected part corresponds to implementation tasks that remain to be carried out by the programmer. These tasks are represented in the code by the presence of stubs. The position of each stub in the code implicitly determines the syntactic class associated with that stub. The implied syntactic class is a characterization of the implementation task in syntactic terms.

A variety of metrics have been developed by software engineers to assess the structural complexity of completed code, programs that have only a prescribed component. A partial metric is one that is able to compute the contribution of both the prescribed and the projected part in order to produce an overall estimate of structural complexity. It was demonstrated that a number of standard metrics, such as McCabe's, McClure's, and Halstead's, can be extended to measure the complexity of pseudocode programs. These extended metrics are especially good predictors of complexity when the code is less than 75% prescribed.

The partial metric used here is that of *program volume*. Program volume is a prediction of the number of bits required to store the corresponding code segment in computer memory. Both prescribed and projected terms contribute to this value. The contribution of the projected terms is set to be the minimum contribution to the count of operands and operators for a nontrivial substitution of code for the projected term. As a result this prediction is a consistent underestimate of the actual value. The relationships necessary to compute program volume are given in figure 5.

Partial program volume reflects certain limits in the amount of pseudocode added at each refinement step [Reynolds 87b]. Results from a study of numerous refinement examples taken from the software engineering literature suggested that the addition of a new pseudocode refinement added on the average 122.79 bits to the partial program volume with a standard deviation of 33. In the generation of the example aggregates in this paper the value was set to the average plus one standard deviation, 155 bits.

A second class of metrics was developed to measure the change in subgoal complexity as the result of a refinement. These metrics are termed *refinement metrics* and measure language support for refinement decisions [Reynolds 88b]. In particular, they are used to estimate the decision-making effort associated with the implementation of a program stub into completed code.

Associated with each term is its identifier class which corresponds to a particular class of nonterminals in the grammar for the target language. It is also possible to associate an index of syntactic complexity with each identifier class. This index is helpful in isolating trends in the selection process for projected terms.

The index used here corresponds to the number of productions in the grammar for the target language that are needed to express that identifier class in terms of the language's terminal symbols. The index, *refinement*

depth, is defined as the maximum number of substitutions, without cycles, that are needed to replace a nonterminal on the right hand side of a grammar's production rule that contains the identifier class on the left hand side. If the nonterminal is on the left hand side of more than one production then the maximum over the set of relevant productions is taken. The index is called refinement depth since it reflects the number of productions required to express the most complex subtask associated with the implementation of a stub of a given nonterminal class.

Refinement depth has been shown to be an important factor in predicting the structure of the code aggregate added to a pseudocode program [Reynolds 87a]. In particular there appears to be a limitation on the extent to which a stub of a given nonterminal will be decomposed into sub-tasks. That is, the distance between the depth for the parent and each of its children seldom exceeds a certain value. This value is always much less than the maximum difference between nonterminal classes in the language. Constraints on the difference between the parent stub and its children reflect a gradualism in the problem solving process.

The second metric constraint placed upon the aggregates produced here is that the parent stub replacing the aggregate in the pseudocode must not be too much larger than its children. That is, the difference between the refinement depth of the parent class and the stubs in the aggregate should be less than a given amount. Here, we discuss aggregates produced using values of 49 and 26. The first value corresponds to the refinement depth for a *statement* and the other to that of an *expression*.

3.2 Generating the Hierarchy of Aggregates

The basic approach used here is to conduct a branch and bound search through possible paths from the target code to a root node. The search is constrained by the values input for refinement depth and partial program volume. Figure 6 presents the result of partitioning the simple report into aggregates using a refinement depth of 49 and a partial program volume of 155. Each bracketed portion of code is indexed by the order in which it was produced.

The aggregation process is performed on the token stream generated from the target code. That is, the characters in the source code are grouped into collections that correspond to reserved words, symbolic names, and other code components. The token stream is a sequence of these tokens produced by scanning the code from top to bottom. Every symbolic name, such as CLEAN_UP, encountered in the scan is associated with a nonterminal class and its associated refinement depth. Initially all symbolic names in the token stream for the target code are associated with the *identifier* class.

Once the token stream is produced the following aggregation procedure is executed:

REPEAT

1. Find the symbolic name associated with the lowest syntactic class. If there is more than one, select the one closest to the end of the token stream. This is the current "seed" for the aggregate.

2. WHILE
   [the partial program volume for the current aggregate is less than the given upper bound]
   AND
   [(the maximum refinement depth for the bracketed syntactic structures minus the minimum refinement depth for the bracketed syntactic structures) is less than or equal to the given upper bound for refinement depth]
   DO
   a. Adjust the top "bracket" by moving the scan to include the preceding token in the stream.
   b. IF [the token is a reserved word that requires another to be syntactically correct, (such as begin), and the other term is currently not in the aggregate (e.g. end)]
      THEN [Adjust the bottom bracket downwards to include the additional term.]
   END WHILE

3. Remove the bracketed code segment and replace it with a new symbolic name. The syntactic class of that name is the nonterminal of highest refinement depth associated with a syntactic structure in the segment. This name corresponds to a stub representing the segment that was removed.

UNTIL (top of token stream reached)

The process can be illustrated using the example code. First, CLEAN_UP is determined to be the seed since it is the closest symbolic name to the end of the program. Next the scan moves the top bracket up to when without encountering a problem. The discovery of when activates the extension of the bottom bracket to pick up the rest of the when action. At this point the difference between the refinement depth values for the largest (*exception-handler*) and smallest (*identifier*) syntactic structures exceeds the refinement depth bound used here of 49. As a result the bracketing process is terminated for the aggregate. The bracketed code is removed and replaced by a unique stub name associated with the aggregate. The process then continues until all aggregates are produced.

3.3 Preparing the Hierarchy of Aggregates for Chunking

Each aggregate, excluding the root, is referenced by a stub in another aggregate. These relationships can be expressed by the tree given in figure 7. Each aggregate corresponds to a node in the tree. The aggregate is labeled by a unique stub name based on the general syntactic class for that node. The integer in the upper right hand corner represents the order in which the aggregate was generated. Note that in our approach all of the leaf nodes are produced prior to the interior nodes.

The number in the lower right hand corner indicates the aggregate's independence from other aggregates. The independence is computed in the following fashion. For each symbolic name in the aggregate's symbol table, divide the number of occurrences in the aggregate by the total number of occurrences in the program. The greater this ratio, the more localized the term's use is in relation to that aggregate. The average of these ratios over all symbolic names in the aggregate is an indication of syntactic independence. Note that the values tend to increase from the leaf aggregates to the root as a natural result of the fact that higher level modules contain more stubs, and all of these stubs are unique occurrences.

The success of the partitioning process described earlier in producing aggregates that exhibit sufficient syntactic independence to facilitate chunking is measured in terms of these values. Since the chunking process described in the next section begins at the leaf aggregates and propagates upward, it is important that the leaf aggregates possess sufficient syntactic independence to allow the process to begin. The rule of thumb used here is that a leaf aggregate should have a ratio in excess of .50 in order to be a viable candidate for chunking. Additionally, the percentage of leaf aggregates that are viable candidates must exceed 50% in order for the process to proceed. If the decomposition does not satisfy this criteria, the metric constraints can be readjusted. This possibility was indicated by a loop from the acceptance checker back to the aggregate generator in figure 2. This search process can be phrased as hill climbing but will not be discussed in

detail here.

Once the hierarchy of aggregates is deemed acceptable it is transformed into a frame-based representation. A frame-based approach to the representation of planning knowledge is employed by Barstow [Barstow 79] and Rich and Waters [Rich 88]. Figure 8 presents a schematic description of information present in the structure for the example. Links in the frame-based representation correspond to each aggregate's attributes. The *is_a* link describes the most specific syntactic class of each aggregate, the *has_prescribed_part* link describes the associated set of tokens, and the *has_projected_part* link describes the stubs yet unrefined.

Note that in the frame hierarchy construction, the aggregates were reordered at each level so that the leftmost child always has the highest syntactic independence measure. In case of a tie, the order corresponds to that in which the aggregates were generated. This reordering facilitates the chunking or abstraction activity to be performed next.

4.0 CHUNKING THE HIERARCHY OF AGGREGATES

In a traditional compiler the semantic processing routines perform the transformation of the source code into target code or some other intermediate representation. The chunking activity described here is analogous in the sense that it is driven by a set of semantic routines whose job is to convert candidate aggregates into a more abstract representation. Each semantic routine is a specialist in generalizing aggregates for a specific syntactic class. The semantic routines are a function of the granularity of the aggregation process as dictated by the metric constraints used. In our example the majority of the aggregates produced are classes of statements. This is reasonable since the refinement metric constraint used corresponds to the depth for the *statement* syntactic class in Ada. The chunking of this example primarily utilizes semantic routines from the statement class.

Figure 9 shows the aggregates produced by reducing the granularity in metric terms. The refinement depth was reduced to 26, the value for an *expression* in Ada. This halving of the depth effectively doubled the number of code aggregates produced. Note that now the predominant syntactic class is that of the *simple-statement*. In this case a much smaller subset of semantic routines are used, and the generalization performed at a lower level. The ordered hierarchy of aggregates is given in figure 10. Due to the size of the tree here, each aggregate is labeled by the order in which it was produced. To the right of each node is the indication of the aggregate's syntactic independence. In this case, 17 out of the 30 leaf aggregates possess acceptable ratios of independence. This partitioning is also an acceptable one for chunking in the system.

The chunking process performed here is expressed as a post-order traversal of the aggregate hierarchy. The procedure essentially searches each node in post-order, first examining the independence ratio to see if it is acceptable. If the index exceeds the current lower bound, .50, then a semantic routine of the same syntactic class is selected to perform a chunking activity. The results of the chunking activity are twofold. First, modifications to the frame describing the current aggregate are made. Second, if the changes to the aggregate require changes to other parts of the program, a propagation request is placed on a semantic stack relative to specific program structures (e.g. variables).

Once the generalization phase is concluded for an aggregate, the semantic stack is checked to see if there are any propagation messages from other nodes that relate to symbolic names present in the current aggregate. If there is a pertinent request, a semantic routine is called to perform the required propagation activity. Ordering the nodes at each subtree level in inverse order of their syntactic independence allows the process described above to be simplified somewhat. If a node at a given subtree level fails to meet the independence criteria, then all the siblings to its right are involved only in the propagation process.

4.1 Chunking Activities

The chunking activities are performed by a collection of semantic routines, where each is associated with a given syntactic class. In general, the more complex the class, the more complex actions of the associated semantic routine(s) are. Currently PM includes only semantic routines for simple statements. The goal of the project is to build up a collection of routines from the bottom up, beginning with the most basic. The absence of semantic routines to deal with all generated classes of aggregates places another constraint on the performance of a successful decomposition. A successful decomposition must now produce a sufficient number of independent aggregates that can be handled by the present collection of semantic routines. The percentage of leaf nodes that satisfy the above constraint for a decomposition is termed its *hit ratio*.

In the first decomposition example, with a refinement depth of 49, only two of the eight leaf aggregates (numbers 3 and 8) are members of the *simple-statement* class. In the second decomposition example, with a refinement depth of 26, 10 of the 30 leaf aggregates have an independence index greater than .50 and consist of a single statement. Therefore, the second decomposition has a hit ratio of .30 as opposed to .25 for the first decomposition. As a result, the second decomposition is more likely to be generalized via the current set of semantic routines, and it is the one that will be used here.

The activities for an example action routine are described in pseudocode below for a simple assignment statement. It is assumed that the aggregate has already satisfied the conditions necessary for chunking.

IF right-hand side = literal THEN

1. Replace the current value in the aggregate's *name* slot with the name of the identifier on the assignment's left hand side.
2. Set the contents of the *has_prescribed_part* slot to nil.
3. Set the contents of the *has_prescribed_part* slot to the identifier on the statement's left hand side.
4. Place a Fill_Stub request on the semantic stack for the identifier [e.g. Fill_Stub(identifier)]. Fill_Stub requests will propagate up to the original block inwhich the variable is defined. It is then added to that aggregate's *Fill_Stub* slot. When the refinement process encounters the aggregate, requests are made for information about each item in the Fill_Stub list.
5. Place a default request on the semantic stack. This will propagate up to the aggregate containing the identifier's definition. The default request's form isDefault (identifier, literal). This information is added to the aggregate's *default* slot.

Semantic routines that handle other kinds of simple statements such as procedure calls have also been designed. Figure 11 shows how a description of the original Ada target code from Rich is chunked using the second decomposition and the current set of semantic routines at the simple statement level. More complicated abstractions, such as enumerated data types, have not been recovered here. Currently work is underway to add such features.

5.0 CONCLUSIONS

This paper has described the current status of the PM project. The system is written in Lisp and runs on a TI Explorer. The metrics-based approach

that underlies the design suggests a systematic way to approach chunking on a large scale and in a language-independent framework.

Future goals of the project are:

1. To expand the set of semantic routines associated with the chunking of Ada code.

2. To allow code from other languages as well. Currently the system is able the compute the partial metrics and refinement metrics for C, Pascal, OPS5 and Prolog.

BIBLIOGRAPHY

[Barstow 79] D. R. Barstow. *Knowledge-based Program Construction.* Elsevier-North Holland, New York, NY 1979.

[Campbell 88] M. S. Campbell. *Chunking as an Abstraction Mechanism.* Carnegie-Mellon University, Computer Science Department. Tech. Rept.-CMU-CS-88-116. February 1988.

[Johnson 87] W. L. Johnson and E. Soloway. PROUST: An automatic debugger forPascal programs. In *Artificial Intelligence and Instruction: Applications and Methods.* G. Kearsley (Ed.) pp. 49-67. Addison-Wesley,Reading, Mass., 1987.

[Kant 81] E. Kant and D. R. Barstow. The Refinement Paradigm: The Interaction of Coding and Efficiency Knowledge in Program Synthesis. *IEEE Transactions on Software Engineering,* SE-7:5, pp. 458-September 1981.

Laird, P. S. Rosenbloom, and A. Newell. Chunking R - The Anatomy of a General Learning Mechanism. *Machine Learning,* 1(1),pp. 11-46, 1986.

[Letovsky 86] S. Letovsky. Cognitive Processes in Program Comprehension. In*Proceedings of the First Workshop on Empirical Studies of Programmers.* E. Soloway and S. Iyengar (Eds.) Ablex Publ. Corp,New Jersey, pp. 58-79, 1986.

[MacLennan 84] B. MacLennan. Simple metrics for programming languages. *Infor-mation Processing and Management,* 20(1/2), pp. 209-221, Jan. 1984.

[Miller 56] G. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review,* 63(2), 1956.

[Reynolds 84] R.G. Reynolds. Metrics to Measure the Complexity of Partial Programs *Journal of Systems and Software,* 4(1), pp. 75-91,April 1984.

[Reynolds 86] R. G. Reynolds. PMS: An Inference System to Monitor the Stepwise Refinement of ADA Pseudocode. *IEEE Expert,* 2(4), pp.43-49,Winter 1986.

[Reynolds 87a] R. G. Reynolds. A Metrics-Based Model for Reasoning about Pseudocode Design. *Journal of Information and Software Technology,* 29(9), pp. 497-502, November 1987

[Reynolds 87b] R. G. Reynolds. The Partial Metrics System: Modeling the Stepwise Refinement Process using Partial Metrics. *Communications of the ACM,* 30(11), pp. 956-963, November 1987.

[Reynolds 88a] R. G. Reynolds. A Metrics-Driven Approach to the Automatic Acqusition of Software Engineering Knowledge. In *Proceedings of the Workshop on Automating Software Design. AAAI-88,,* St. Paul, MN,pp. 156-158, August 1988.

[Reynolds 88b] R. G. Reynolds and J. I. Maletic. *Refinement Metrics: Measures of decision-making effort in the stepwise refinement process.* Wayne State University, Computer Science Department. Research Report. November 1988.

[Rich 88] C. Rich and R. C. Waters. The Programmer's Apprentice: A Research Overview. *IEEE Computer,* 21(11), pp. 10-25, November 1988.

[Rist 86] R. S. Rist. Plans in Programming: Definition, Demonstration, and Development. In *Proceedings of the First Workshop on Empirical Studies of Programmers.* E. Soloway and S. Iyengar (Eds.) Ablex Publ. Corp, New Jersey, pp. 28-47, 1986.

[Rosenbloom 83] P. S. Rosenbloom. The Chunking of Goal Hierarchies. Carnegie-Mellon University, Computer Science Department. PhD Thesis, 1983.

[Roy 88] S. Roy and J. Mostow. *Parsing to Learn Fine Grained Rules.* Rutgers University, Department of Computer Science. AI/Design ProjectWorking Paper 69. June 1988.

[Simon 69] H. A. Simon. *The Sciences of the Artificial.* M.I.T. Press,Cambridge, Mass., 1969.

[Simon 86] H. A. Simon. Whether software engineering needs to be artificially intelligent. *IEEE Transactions on Software Engineering,* SE-12:7, pp. 726-732, July 1986.

[Soloway 88] E. Soloway and F. Detienne. *An Empirically-Derived Control Struc-ture for the Process of Program Understanding.,* Rapports de Recherche No. 886, B.P. 105, 78153 Le Chesnay Cedex, France, Aout 1988

[Tambe 88] M. Tambe and A. Newell. *Why Some Chunks Are Expensive.* Car-negie-Mellon University, Computer Science Department. Tech. Rept. CMU-CS-88-103. January 1988.

[Winston 84] P. H. Winston. *Artificial Intelligence.* Addison-Wesley, Reading, Mass., 1984.

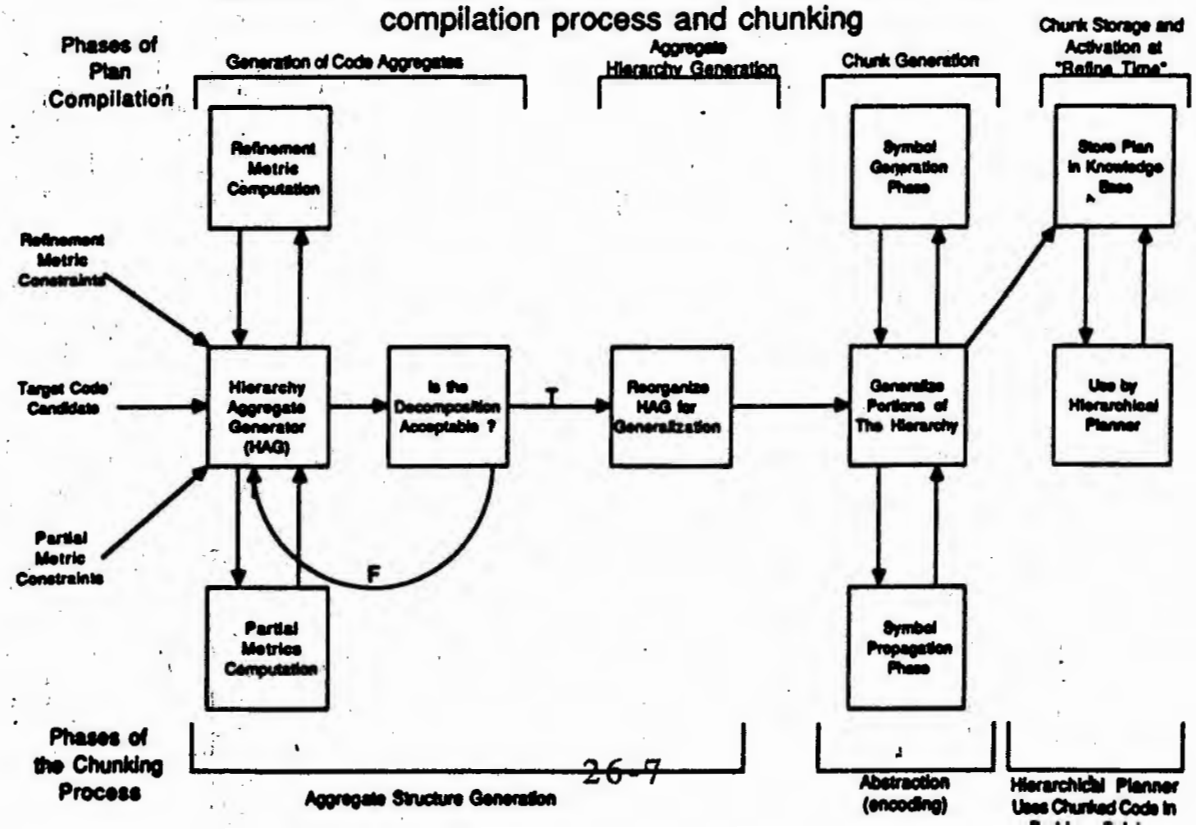
```
with CALENDAR, FUNCTIONS, TEXT_IO;
use CALENDAR, FUNCTIONS, TEXT_IO;
elicit SIMPLE_REPORT is
  primary roles ENUMERATOR, PRINT_ITEM, SUMMARY;
  described roles FILE_NAME, TITLE, ENUMERATOR,
                  COLUMN_HEADINGS, PRINT_ITEM, SUMMARY;
  comment "prints a report of {the input data of the enumerator}";
  constraints
    DEFAULT({the file_name}, "report.txt");
    DERIVED({the line_limit},
              66 - SIZE_IN_LINES({the print_item})
                 - SIZE_IN_LINES({the summary}));
    DEFAULT({the print_item},
              CORRESPONDING_PRINTING({the enumerator}));
    DEFAULT({the column_headings},
              CORRESPONDING_HEADINGS({the print_item}));
  end constraints;
  use INT_IO;
  CURRENT_DATE: constant STRING := FORMAT_DATE(CLOCK);
  DATA: ( );
  REPORT: TEXT_IO.FILE_TYPE;
  TITLE: STRING(1..{ });
  procedure CLEAN_UP is
  begin
    SET_OUTPUT(STANDARD_OUTPUT); CLOSE(REPORT);
  exception when STATUS_ERROR => return;
  end CLEAN_UP;
begin
  CREATE(REPORT, OUT_FILE, {the file_name});
  DATA := {the input data of the enumerator};
  SET_OUTPUT(REPORT);
  TITLE := {the title};
  NEW_LINE(4); SET_COL(20); PUT(CURRENT_DATE); NEW_LINE(2);
  SET_COL(13); PUT(TITLE); NEW_LINE(60);
  while not {the empty_test of the enumerator}(DATA) loop
    if LINE > {the line_limit} then
      NEW_PAGE; NEW_LINE; PUT("Page: ");
      PUT(INTEGER(PAGE - 1), 3); SET_COL(13); PUT(TITLE);
      SET_COL(61); PUT(CURRENT_DATE); NEW_LINE(2);
      {the column_headings}({CURRENT_OUTPUT, modified});
    end if;
    {the print_item}({CURRENT_OUTPUT, modified},
                    {the accessor of the enumerator}(DATA));
    DATA := {the step of the enumerator}(DATA);
  end loop;
  {the summary}({CURRENT_OUTPUT, modified});
  CLEAN_UP;
exception
  when DEVICE_ERROR | END_ERROR | NAME_ERROR
       | STATUS_ERROR =>
    CLEAN_UP; PUT("Data Base Inconsistent");
  when others => CLEAN_UP; raise;
end SIMPLE_REPORT;
```

Figure 1.
Simple Report Cliche for the
Programmers Apprentice taken from [Rich 88]

```
with CALENDAR, FUNCTIONS, MAINTENANCE_FILES, TEXT_IO;
use CALENDAR, FUNCTIONS, MAINTENANCE_FILES, TEXT_IO;
procedure UNIT_REPAIR_REPORT is
  use DEFECT_IO, REPAIR_IO, UNIT_IO, INT_IO;
  CURRENT_DATE: constant STRING := FORMAT_DATE(CLOCK);
  DEFECT: DEFECT_TYPE;
  REPAIR: REPAIR_TYPE;
  REPAIR_INDEX: REPAIR_INDEX_TYPE;
  REPORT: TEXT_IO.FILE_TYPE;
  TITLE: STRING(1..33);
  UNIT: UNIT_TYPE;
  UNIT_KEY: UNIT_KEY_TYPE;
  procedure CLEAN_UP is
    begin
      SET_OUTPUT(STANDARD_OUTPUT);
      CLOSE(DEFECTS); CLOSE(REPAIRS); CLOSE(UNITS); CLOSE(REPORT);
    exception
      when STATUS_ERROR => return;
    end CLEAN_UP;
begin
  OPEN(DEFECTS, IN_FILE, DEFECTS_NAME); OPEN(REPAIRS, IN_FILE, REPAIRS_NAME);
  OPEN(UNITS, IN_FILE, UNITS_NAME); CREATE(REPORT, OUT_FILE, "report.txt");
  loop
    begin
      NEW_LINE; PUT("Enter UNIT Key: "); GET(UNIT_KEY);
      READ(UNITS, UNIT, UNIT_KEY);
      exit;
    exception
      when END_ERROR => PUT("Invalid UNIT Key"); NEW_LINE;
    end;
  end loop;
  TITLE := "Report of Repairs on Unit " & UNIT_KEY;
  SET_OUTPUT(REPORT);
  NEW_LINE(4); SET_COL(20); PUT(CURRENT_DATE);
  NEW_LINE(2); SET_COL(13); PUT(TITLE); NEW_LINE(60);
  READ(UNITS, UNIT, UNIT_KEY); REPAIR_INDEX := UNIT.REPAIR;
  while not NULL_INDEX(REPAIR_INDEX) loop
    READ(REPAIRS, REPAIR, REPAIR_INDEX);
    if LINE > 64 then
      NEW_PAGE; NEW_LINE; PUT("Page: "); PUT(INTEGER(PAGE - 1), 3);
      SET_COL(13); PUT(TITLE); SET_COL(61); PUT(CURRENT_DATE); NEW_LINE(2);
      PUT("    Date       Defect    Description/Comment"); NEW_LINE(2);
    end if;
    READ(DEFECTS, DEFECT, REPAIR.DEFECT);
    PUT(FORMAT_DATE(REPAIR.DATE)); SET_COL(13); PUT(REPAIR.DEFECT);
    SET_COL(26); PUT(DEFECT.NAME); NEW_LINE;
    SET_COL(22); PUT(REPAIR.COMMENT); NEW_LINE;
    REPAIR_INDEX := REPAIR.NEXT;
  end loop;
  CLEAN_UP;
exception
  when DEVICE_ERROR | END_ERROR | NAME_ERROR | STATUS_ERROR =>
    CLEAN_UP; PUT("Data Base Inconsistent");
  when others => CLEAN_UP; raise; end UNIT_REPAIR_REPORT;
```

Figure 3.
Ada target code produced by the refinement of the Simple Report cliche in the Programmers Apprentice taken from [Rich 88]

Figure 2. Relationship between the phases of the PM compilation process and chunking



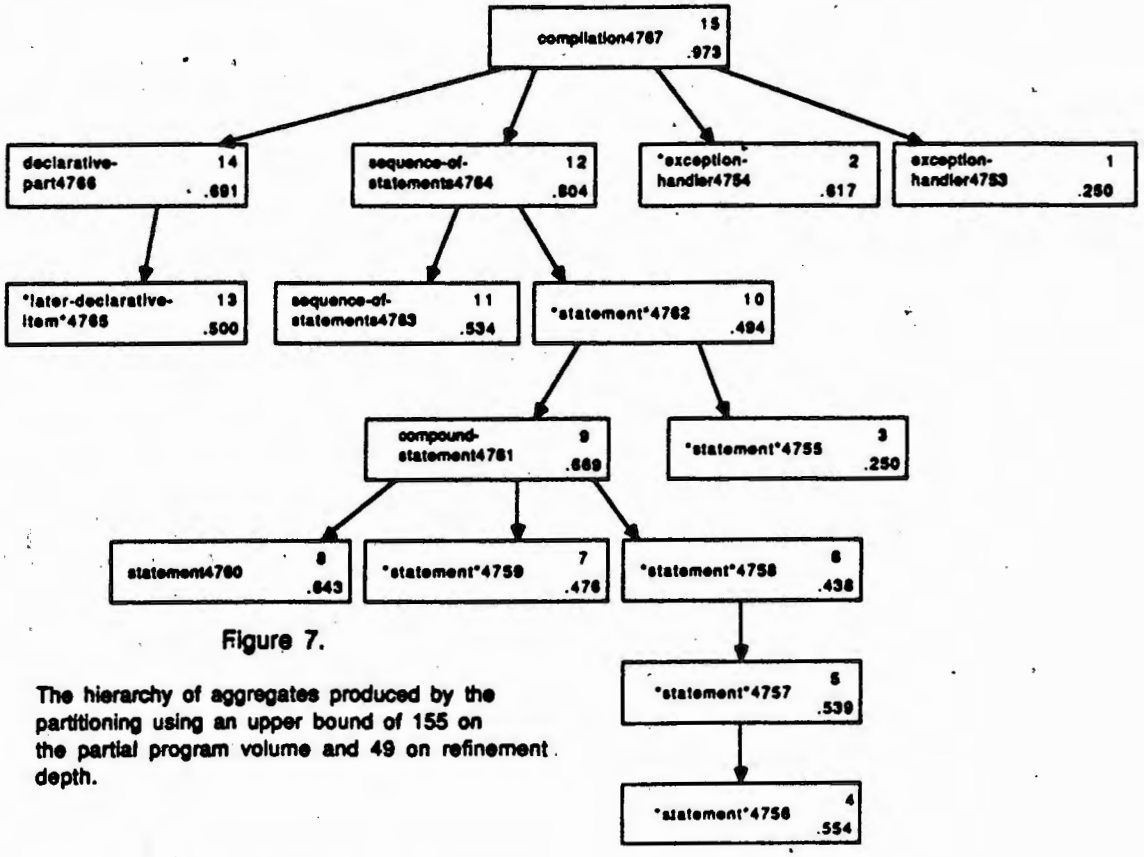


Figure 7.

The hierarchy of aggregates produced by the partitioning using an upper bound of 155 on the partial program volume and 49 on refinement depth.

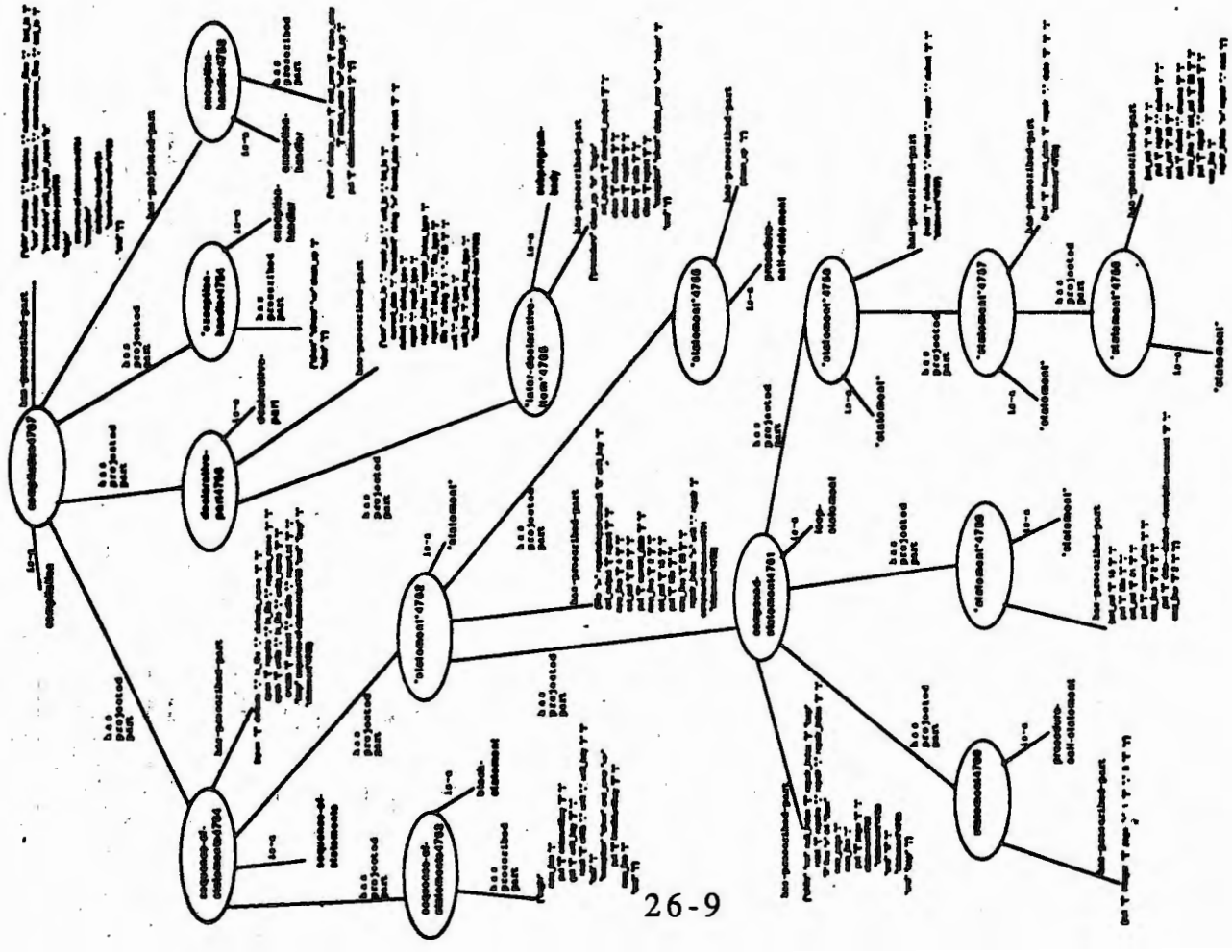


Figure 8.

The hierarchy of aggregates expressed as a frame structure. Note that at each level the aggregates are ordered from left to right in decreasing order of structural independence.

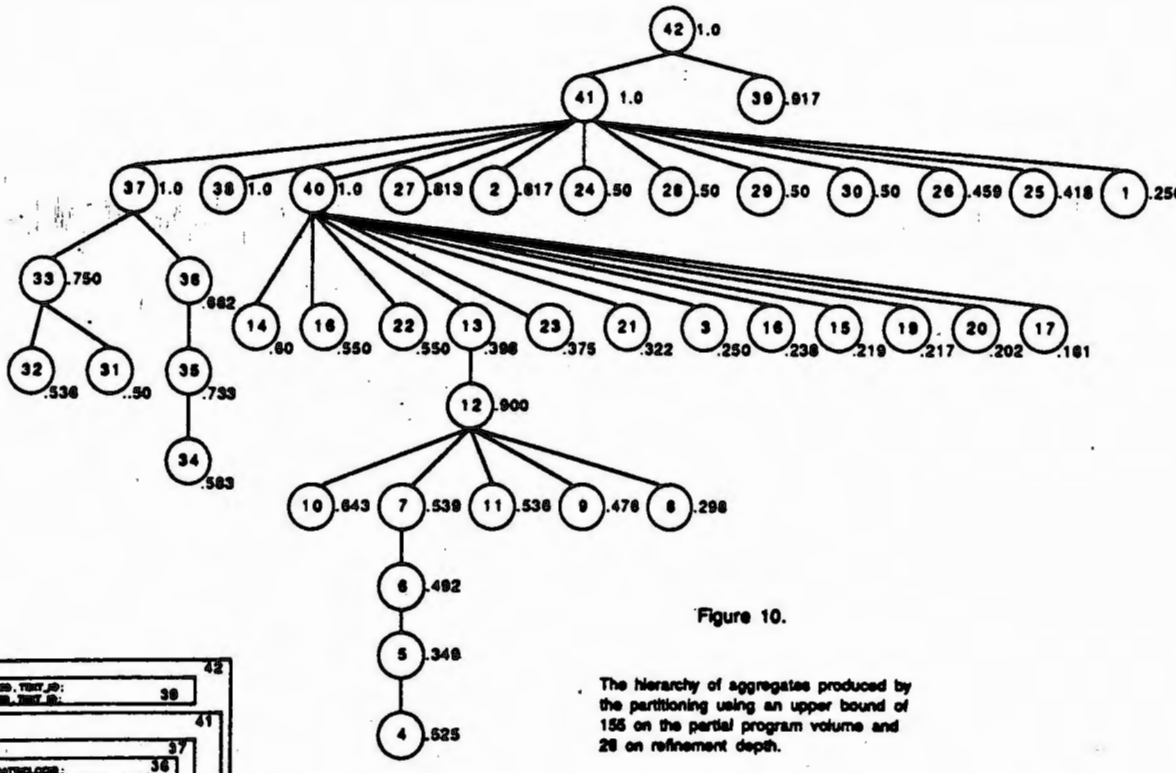


Figure 10.

The hierarchy of aggregates produced by the partitioning using an upper bound of 155 on the partial program volume and 28 on refinement depth.

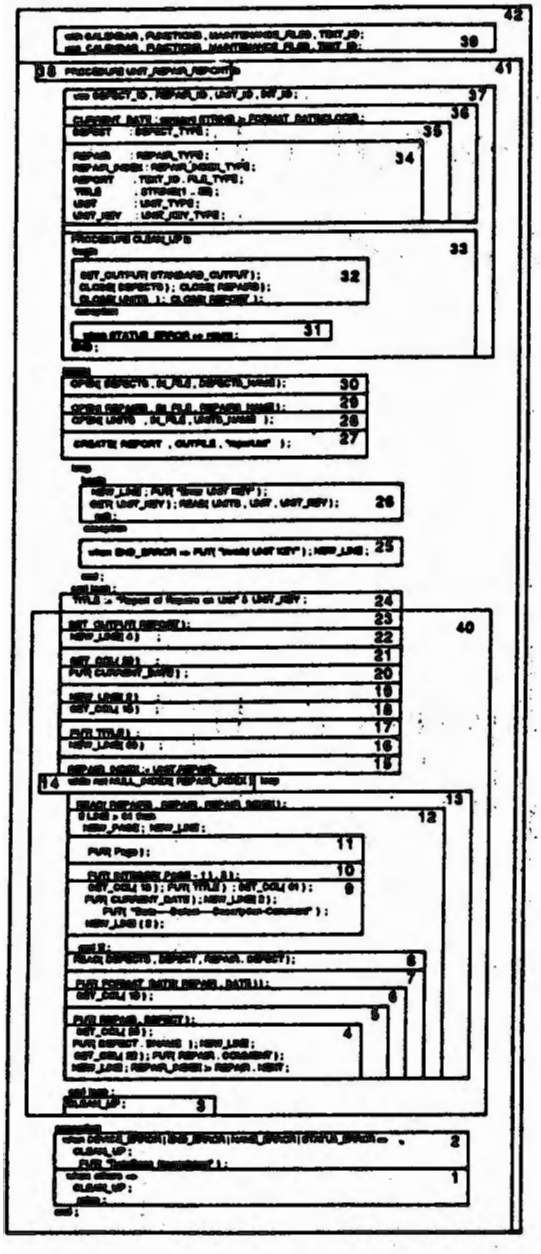


Figure 9.

The partitioning of the simple report program using a bound

```
Fill-stub(/title/)            Fill-stub(/header/)
Fill-stub(/line-length/)      Fill-stub([output-file-spec])

Default(/title/,"Report")     Default([header],"Header")
Default(/line-length/,80)     Default(/output-file-spec/,"report.txt")

with CALENDAR , FUNCTIONS , MAINTENANCE_FILES , TEXT_IO ;
use  CALENDAR , FUNCTIONS , MAINTENANCE_FILES , TEXT_IO ;

PROCEDURE UNIT_REPAIR_REPORT is
  use DEFECT_IO , REPAIR_IO , UNIT_IO , INT_IO ;
  CURRENT_DATE : constant STRING := FORMAT_DATE(CLOCK) ;   DEFECT       : DEFECT_TYPE ;
  REPAIR       : REPAIR_TYPE ;      REPAIR_INDEX : REPAIR_INDEX_TYPE ;
  REPORT       : TEXT_IO.FILE_TYPE ; TITLE        : STRING(1 .. 33) ;
  UNIT         : UNIT_TYPE ;        UNIT_KEY     : UNIT_KEY_TYPE ;

  PROCEDURE CLEAN_UP is
  begin
    SET_OUTPUT( STANDARD_OUTPUT ) ; CLOSE( DEFECTS ) ;
    CLOSE( REPAIRS ) ; CLOSE( UNITS  ) ; CLOSE(/report/) ;
  exception  when STATUS_ERROR => return ;
  END ;

BEGIN
  OPEN( DEFECTS , IN_FILE , DEFECTS_NAME ); OPEN( REPAIRS , IN_FILE , REPAIRS_NAME ) ;
  OPEN( UNITS  , IN_FILE , UNITS_NAME); CREATE( REPORT , OUTFILE ,/output-file-spec/ ) ;
  loop  begin
      NEW_LINE; PUT( "Enter UNIT KEY" ); GET( UNIT_KEY ); READ( UNITS , UNIT , UNIT_KEY ) ;
      exit ;
    exception
      when END_ERROR => PUT( "Invalid UNIT KEY" ) ; NEW_LINE ;
  end ; end loop ;
  /title/; SET_OUTPUT(/report/) ; NEW_LINE( 4 ); SET_COL(20);
  PUT(CURRENT_DATE); NEW_LINE(2); SET_COL(13); PUT(/title/); NEW_LINE(/line-length/);
  while not NULL_INDEX( REPAIR_INDEX ) loop
    READ( REPAIRS , REPAIR , REPAIR_INDEX ) ;
    if LINE > 64 then
      NEW_PAGE ; NEW_LINE ; PUT( Page ) ;  PUT( INTEGER( PAGE - 1 ) , 3 ) ; SET_COL( 13 ) ;
      PUT( TITLE ) ;  SET_COL( 61 ) ; PUT( CURRENT_DATE ) ;
      NEW_LINE( 2 ) ; /header/; NEW_LINE ( 2 ) ;
    end if ;
    READ( DEFECTS , DEFECT , REPAIR . DEFECT ) ;  PUT( FORMAT_DATE( REPAIR . DATE ) ) ;
    SET_COL( 13 ) ; PUT( REPAIR . DEFECT ) ; SET_COL( 20 ) ;  PUT( DEFECT.DNAME  ) ; NEW_LINE ;
    SET_COL( 22 ) ; PUT( REPAIR.COMMENT ) ; NEW_LINE ;  REPAIR_INDEX := REPAIR . NEXT ;
  end loop ;
  CLEAN_UP ;
exception  when DEVICE_ERROR | END_ERROR | NAME_ERROR | STATUS_ERROR => CLEAN_UP ;
           PUT( "DataBase Inconsistent" ) ;
         when others => CLEAN_UP ; raise ;
end ;
```

Figure 11.
Results of chunking the simple report target code using a simple set of semantic routines.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

KNOWLEDGE ACQUISITION AS COGNITIVE MODELING[1]

Valerie L. Shalin
John R. Bloomfield
Honeywell Systems and Research Center
3660 Technology Dr., MN65-2500
Minneapolis, MN 55418
shalin@src.honeywell.com

Peter T. Bullemer
Honeywell Sensor and System Development Center
1000 Boone Ave. N.
Golden Valley, MN 55427

ABSTRACT

We have adopted a cognitive modeling approach to the problem of acquiring knowledge from experts for building expert systems. According to this approach, we attempt to capture and represent the content, structure and processes of knowledge underlying expert task performance. In this paper, we describe a knowledge acquisition methodology that is grounded in the experimental methods of cognitive psychology and designed to address the challenges presented by the tacit nature of knowledge underlying expertise. The goals of our approach are influenced by current concepts of knowledge in the cognitive literature, and how knowledge may overcome the constraints of cognitive architecture, such as a limited capacity attentional process. We present some preliminary findings of this methodology applied to the domain of aerial image analysis, and discuss the relationship between the representation of domain knowledge in expert systems and expert human problem solvers.

1.0 INTRODUCTION

The primary source of knowledge for expert systems is the domain expert. Unfortunately, for most domains, and especially those domains involving complex tasks, it is peculiarly difficult to extract much of the knowledge the expert uses. This is because:

- Experts rarely operate at a basic level —they make complex judgments very rapidly;
- The skill acquisition process ensures that the reasoning processes and task-related behaviors used by experts are overlearned and overpracticed (Anderson, 1983; Fitts, 1964; Schneider 1985; Shiffrin & Schneider, 1977). With practice, the sequential reasoning that occurs for a novice is "compiled out" and stored as a "chunk" (Laird, Rosenbloom, & Newell, 1986). The chunk of knowledge is accessed and applied virtually instantaneously. The detailed reasoning that supports the chunk is not easily accessed by well-practiced experts;
- Much of an expert's knowledge is difficult, if not impossible, to articulate. The knowledge may have always been tacit, associational and without a strong causal relationship existing between the premises and the conclusions (Berry, 1987). This may be especially true of domains where the cues are nonsymbolic attributes of objects or complex relationships between multiple objects and their attributes (e.g., Berry & Broadbent, 1984; Lewicki, 1986). Such implicit knowledge is acquired through procedural learning (Nissen & Bullemer, 1987; Reber, Kassin, Lewis & Cantor, 1980; Willingham, Nissen & Bullemer, in press). The expert may have learned the task as an apprentice and never have acquired

[1]Portions of sections 1 and 2 of this paper have appeared in the Proceedings of the 1989 Ergonomics Society Annual Conference, J. R. Bloomfield & V. L. Shalin "Knowledge Acquisition Techniques: Diagnosis and Prognosis". We are grateful to Tim Ayers for programming the model of image analysis discussed in this paper. The work was internally supported by Honeywell Inc.

declarative knowledge of the reasoning underlying its execution. Consequently, the tasks may be passed on from generation to generation without any first principles ever being acquired (Collins, Green & Draper, 1985); and

- There is a "surprising degree of dislocation [...] between conscious, verbally-expressed behavior and the tacit knowledge which guides skilled behavior" (Howarth, 1988).

In spite of these problems, a primary approach to knowledge acquisition for expert systems has been to go and ask experts what they know.

Knowledge acquisition has much in common with the area of psychology known as cognitive modeling[2]. Both address the problem of discovering the knowledge that underlies performance at a level of detail sufficient for simulating the reasoning process. Despite having common goals, cognitive psychology has had less influence on knowledge acquisition than it should. For example, an important methodological flaw in the "just-go-and-ask" approach is the failure to consider the relationship between the expert's task in responding to the inquiries of the knowledge engineer and the expert's normal task. A number of psychological studies have shown that tacit knowledge is triggered by the contextual features of the environment in which it was acquired (Anderson, 1983; Anderson & Ortony, 1975; Hyde & Jenkins, 1969; Tulving & Thompson, 1973). In addition to highlighting such methodological concerns, cognitive psychology provides several theories of knowledge with great relevance to the goals of knowledge acquisition.

We believe that the advantage of a shared perspective between the knowledge acquisition and cognitive modeling communities has advantages that extend beyond a solution to the knowledge acquisition problem itself. We anticipate that a number of near-term expert system aids will be assistants, incapable of performing the complete task without the human expert (Levi, Shalin, & Perschbacher, 1988; Shalin, Perschbacher, & Jamar, 1988). The model of domain knowledge that guides these systems will have to be compatible to some degree with that of the human user, particularly in situations where the user interacts directly with the system at its most primitive level of processing, for example in requesting explanations or modifying code. We have argued this general approach to expert systems elsewhere (Shalin & Boy, in preparation) and note that Harlow, Trivedi, Conners, & Phillips (1986) argue this approach for the image analysis domain addressed in this paper.

In the next section, we discuss some examples of cognitive theories of knowledge and the nature of expertise. Then, we describe some of the existing methodologies for knowledge acquisition. This provides a background for us to introduce our "knowledge sensitive task manipulation" as an approach to the problem of identifying tacit knowledge. In the last section of this paper we describe the preliminary results of an ongoing project to apply knowledge sensitive task manipulation to explore the reasoning and problem solving of expert aerial image analysts.

2.0 PURPOSE OF KNOWLEDGE ACQUISITION

The purpose of our own knowledge acquisition efforts is to develop a computational model of the cognitive processes that support expert task performance. "Computational" refers to the fact that the model mimics the step-wise reasoning process and produces an answer[3]. It is our contention that the essence of a cognitive model of expertise is to account for the successful functioning of a

---

[2]For the purposes of this paper, we distinguish knowledge acquisition (identifying the knowledge used in reasoning) from knowledge engineering (which may include both knowledge acquisition and coding the identified knowledge for processing by the expert system).

[3]Such models are generally implemented using programming languages based on concepts of cognitive architecture (Anderson, 1983; Laird, Rosenbloom, and Newell, 1987). For the purposes of knowledge acquisition, we are not concerned with the particular choice of implementation language here. We do wish to maintain in spirit the constraints they impose on modeling expertise.

limited capacity working memory that is unable to accommodate many tasks in parallel (see Simon, 1969; and VanLehn, 1983, for a related perspective.) Thus: knowledge helps determine which aspects of the task environment shall be addressed first; knowledge helps determine which of several alternative rules of inference shall be tested and applied at any point in the problem solving process; and knowledge helps determine when a sufficient amount of information has been obtained in ill-structured domains to declare the problem solved.

When confronted with the need to acquire expert knowledge in a new domain, our goal is to uncover how these potentially tacit, knowledge/attention related phenomena are instantiated for the new domain. Without ever looking at a single domain specific account, it is clear to us that aerial image analysis must involve knowledge related to at least the following general aspects of human reasoning:

2.1 Hierarchical Class Structure in Object Identification

Rosch (1978) has shown that when performing object recognition tasks, subjects have a preferred level of analysis for object recognition, depending upon the use of the object in later processing. They identified three levels: the *basic* level, with the *superordinate* level above it and the *subordinate* level below it. The level of analysis governs which details of an object are examined for object recognition, so that a basic level analysis does not require the examination of features that are only relevant to subordinate level distinctions. Thus, one goal of knowledge acquisition is to identify the hierarchical class structure in the domain, and the features associated with each class.

2.2 The Role of Context in Object Identification

Context affects how people interpret the objects in a reasoning task. This has been shown in the interpretation of text (Anderson & Ortony, 1975), the posited role of objects in novel construction tasks (Duncker, 1945), and the identification of objects in impoverished or misleading images (Palmer, 1975). Thus, a second goal of knowledge acquisition is to investigate the role of context in interpreting or identifying the objects in the domain.

2.3 Domain-specific Intermediate Subgoals

Larkin (1982) has observed that experts frequently go through a lengthy process of problem elaboration, without ever checking for the specific unknown of a physics problem. This process of forward chaining is probably mediated by the use of "schemas" that match certain abstract problem features. Schemas guide forward chaining to address those problem subgoals that will, in all likelihood, be required for any problem matched by that schema. This has been noted in other areas as well (e.g., decision making, see Klein, in press, and Bloomfield, Peio, Lehman, Masters, & Boettcher, 1989). The use of schemas standardizes the problem solving process, directs the focus of attention to the establishment of relevant subgoals and allows for some modest growth in problem complexity, such as constructions (Greeno, Magone, & Chaiklin, 1979). Accordingly, an important goal of knowledge acquisition is to uncover the "schemas" in the domain, if they exist, and the procedures that are executed automatically when a schema is matched.

2.4 Knowledge for Constructing Coherent Representations in Interpretation Tasks

Many real-world problem solving tasks lack well-defined criteria that indicate when a solution has been achieved (Reitman, 1965; Simon, 1973). Greeno (1979) has suggested that these and other problems are solved when the problem solver has obtained a *coherent representation* of the problem. Thus, another goal in expert knowledge acquisition is to identify the basis on which a

problem is deemed solved, and—when coherence provides this basis—how the sampling of features is contained to avoid exhaustive search.

As a matter of standard practice, we apply our knowledge acquisition methods in a goal directed manner to investigate these and other types of knowledge for a given task domain.

3.0 KNOWLEDGE ACQUISITION

In the previous section we identified some of the *goals* for knowledge acquisition that we can adopt from the cognitive literature on the theories of knowledge underlying expertise. In this section we review some of the *standard approaches* to knowledge acquisition, with particular attention to those approaches that have emerged from the literature on cognitive models of expertise. The following discussion, which is organized to highlight a distinction between dependent measures and the expert's task and task manipulations, will put the "knowledge sensitive task manipulation." approach in perspective. *Dependent measures* are the expert behaviors that we record and analyze. In keeping with Newell & Simon's (1972) definition of a task environment, *tasks* consist of problem statements, solution methods and goal conditions. Dependent measures are sensitive to the process of task performance, and are somewhat independent of tasks. For example, one might measure the performance of the aerial image analysis task with verbal protocols, normal task behaviors, or some derived measure on these, such as the clustering or ordering in time of particular responses or behaviors.

3.1 Dependent Measures[4]

Verbal Responses—The predominant dependent measure for knowledge acquisition is the expert's verbal response. The expert's comments are often taped as verbal protocols. The critical dimension of the verbal response is the degree of introspection it requires of the domain expert. Following Olson & Rueter (1987), direct methods require the expert to introspect and articulate on the knowledge underlying task performance. Indirect methods do not rely on the expert's ability to articulate his knowledge, but depend on the observers analysis of other types of behavioral data, such as those mentioned below. The drawbacks with verbal protocols are: 1) acquiring them can be very time-consuming and the process yields large amounts of data that are in a non-standard format and are difficult to analyze, 2) the act of discussing or explaining his/her approach may actually alter the expert's normal performance (Ericsson & Simon, 1986), and 3) neither direct nor indirect methods are well-suited to the acquisition of implicit/tacit knowledge: The direct method requires the expert to introspect about something he or she probably cannot access (Nisbett & Wilson, 1977; Osgood, 1953). The indirect method *alone* does little to ensure an adequate investigation of tacit knowledge.

Normal Task Performance—With an ethological approach, which Waterman (1986) described as the on-site observation method, the knowledge engineer merely observes the expert performing his/her task. The engineer is a passive observer and, as a result, is likely to have a less inhibiting or interfering effect on the expert's performance compared to verbal protocols. Langley and Ohlsson (1984) and VanLehn and his colleagues (VanLehn & Garlick, 1987; Kowalski & VanLehn, 1988) have been investigating automated approaches to the problem of inducing an individual's knowledge base purely from their responses to a carefully crafted problem set in the

---

[4] We have by no means exhausted the set of potential dependent measures in this section. For example, we know of attempts to use eye movements and brain waves as indications of thought processes. We have included in this section measures where there are reasonably strong causal mechanisms to explain the relationship between the dependent measure and the internal thought processes. A further consideration of eye movement and brain wave data is the sensitivity of the measure relative to the question being asked: These very sensitive measures may be too variable and therefore uninformative when the research questions are gross and exploratory, but quite appropriate when some very specific hypothesis is being tested.

domain of subtraction. The main drawback of their methods is that they only apply to very simple task domains. A recent development is to record via sensors all of the details of the changing task environment along with the expert actions as he or she executes the task of interest. The resulting record is then amenable to semi-automated knowledge acquisition (Shalin, Levi, and Perschbacher, 1989). The main drawback to the observation of normal task performance, if it is the only source of observations, is that the knowledge engineer can only analyze the expert's performance, and cannot get at his/her thought processes or determine which are the crucial observations.

Clustering and Ordering of Responses—This dependent measure is derived from the two mentioned previously. The expert provides some stream of behavior (verbal or otherwise) which is grouped over time as a task is performed. Chase & Simon (1973) and Reitman (1976) discovered the patterns that expert board game players use by examining how many pieces an expert could reproduce from an existing position following a single glance at the board. McKeithen, Reitman, Ruter, & Hirtle (1981) used a similar method to examine the structure of a written computer program. In a somewhat related variant, Chi, Glaser, & Rees (1981) asked experts to group problem statements as members of the same or different categories of problems. Note that for the cases cited here, response clustering evidence is obtained with a change to the expert's normal task. In keeping with our earlier criticism of standard approaches to knowledge acquisition, one problem with the clustering and ordering measures is that it is not certain that the observations obtained from this artificial task have anything to do with real task performance. We return to this issue below where we explicitly consider the use of task manipulations in knowledge acquisition.

3.2 The Expert's Task and Knowledge Acquisition Methodologies

Dependent measures define only half of an approach to knowledge acquisition. No matter which measure is used, the observations are always obtained within the context of some task, even if the task is only implicit (e.g., tell me what you do). Johnson (1984) indicates the importance of task veridicality on the authenticity of the data. In addition, it is our belief that the task conditions can be manipulated to enhance the completeness of the resulting conclusions about the knowledge used in the domain of interest. We describe below some of the standard knowledge acquisition methodologies and how they impact the expert's task, and turn to the question of task manipulations in a later section. The focus of the following section is on the degree of task veridicality, and some of the risks associated with each approach. We refer the reader to Olson et al. (1987), Hoffman (1987), and Waterman (1986) for a more complete presentation of each method and their relationships to one another.

Unstructured Interviews—In an unstructured interview, the knowledge engineer asks a series of semi-spontaneous questions about the expert's task. Given the minimal preparation required by the knowledge engineer, the unstructured interview method only produces good information if an enormous amount of time is spent in multiple sessions, or if the expert fortuitously happens to produce the information that the engineer needs. Otherwise with only a few sessions, the method will only produce incomplete information, with the significant elements of information are obscured by superficial observations.

Structured Interviews—The key to a structured interview is that the knowledge engineer has some preconceptions about the nature of the expert's knowledge base and uses this preconception to explicitly guide the exchange with the expert. For example, a knowledge engineer might prepare for a structured interview by producing a strawman knowledge base from an analysis of task descriptions, technical manuals, or the output of an unstructured interview. Then, in conducting the structured interview, the knowledge engineer systematically goes through the knowledge base, while the expert points out if, and where, structural changes need to be made, and indicates any necessary additions, deletions, or qualifications. We include in this category the many knowledge acquisition tools (such as , KLAMSHELL, by Edman, Cochran, & Hadden, 1988; Acquinas by

Boose & Bradshaw (1987), and learning apprentice systems such as LEAP (Mitchell, Mahadevan, & Steinberg, 1985), and TIERESIAS (Davis, 1979).

The use of the structured interview should result in a shorter, more productive, knowledge acquisition process than when the interviews are unstructured. However, the domain expert is often a poor critic, especially when understanding the syntax of the knowledge base is required. Second, when knowledge acquisition tools take the expert out of the standard task environment, there are always the risks of acquiring irrelevant "knowledge" and of failing to acquire strategic and procedural knowledge that is needed to execute the task.

On-the-Job Commentary (on-line protocols)—With this method, the domain expert provides a commentary as he/she performed his/her tasks, and the knowledge engineer subsequently analyzes it. When on-line verbal protocols are obtained while the expert executes his or her normal task, the knowledge acquired is more likely to be related to the task of interest than data obtained by an undirected "data dump" and post hoc expert rationalizations (Ericsson & Simon, 1986). Like unstructured interviews, however, there may be few cues that highlight the significant parts of the protocol.

Simulation—One variant of on-the-job commentary (and structured interviewing as well) is obtained by using archival scenarios played in interruptible real-time. Because it is possible to interrupt at any point, and question the expert, the knowledge engineer may be able to acquire information on some subtle point that might have otherwise been missed. Moninger & Stewart (1987) used an archival simulation approach in a study of the expertise of weather forecasters. Interestingly, the main strength of this approach, that it gives the engineer the opportunity to gain detailed knowledge from the expert at any point in the task, is likely to be its main weakness, also. Frequent stoppages for discussion with the knowledge engineer will interrupt the task flow, and may taint the findings.

Coaching—A second variant of on-the-job commentary is based on coaching (Waterman, 1986). The expert is asked to describe his/her task to a novice who must then execute the task. A similar methodology has been used in cognitive psychology to uncover the knowledge that young children have about the counting process. As we have commented, a primary drawback of this approach is the relevance of the stated procedure to the method actually used by the expert.

Expert-Novice—Most of the above methods can incorporate expert-novice comparisons. An analysis of the differences between the verbal protocols of the experts and novices performing their normal task, can assist the knowledge engineer in determining which of the insights provided by the experts are likely to be most critical for performing their tasks efficiently. Larkin, McDermott, Simon & Simon (1981) used this approach, obtaining verbal protocols from the experts and novices whose task was to solve physics problems. This method introduces an issue pursued in more detail in the next paragraph; that is, the importance of the specific scenarios or problems within a knowledge acquisition task. Chi, Feltovich, & Glaser (1981) have noted that experts provide very little evidence of their step-wise reasoning processes when they are solving well-practiced problems. Thus, expert-novice comparisons using problems that novices can solve may be more revealing about the nature of novice performance than expert performance.

Tough Cases—To identify the step-wise reasoning process of experts, a knowledge engineer might identify particularly challenging aspects of the expert's task. The knowledge engineer then proceeds with the on-the-job commentary method using these challenging cases. Hoffman (1984) used the method of tough cases to explore a terrain analysis task using high-altitude photographs. The difficulty of a challenging case prompts the expert to reason with more basic first principles. There are, however, drawbacks: in many domains, challenging cases are relatively rare, and will not cover many aspects of the expert's task. Some challenging cases may be so atypical of the expert's tasks that they provide misleading information. While the method of tough cases forces

the expert to reason at a more basic level of analysis on problem components, it may also eliminate the very abstract representational knowledge with which we are concerned, for example, knowledge about problem features that map onto schemas or problem categories.

Variations in the Task Environment—Knowledge acquisition techniques involving changes in the task environment are intrusive, with associated risks. But, they may prompt the expert into articulating some pieces of knowledge that otherwise would have been unobtainable (Chi et al., 1982). Hoffman (1984) used two variations of the task environment in his study of expert terrain analysis, both of which made the normal task harder. For the first variation, he limited the amount of background information given to his photointerpreters, while in the second, he greatly reduced the length of time they were allowed to view aerial photographs. To obtain knowledge about familiar patterns in chess, Chase et al. (1973) investigated the chess-board reproduction task. To obtain representational knowledge about problem categories, Chi et al. (1982) asked subjects to group problem statements as members of problem categories in an effort to obtain tacit representational knowledge.

3.3 Knowledge Sensitive Task Manipulation

We have taken the idea of altering the task environment explored in these earlier studies, and developed from it an approach in which the task conditions are systematically manipulated, on a trial-to-trial basis. In each case, the intent is to systematically interfere with the expert's task, with the effect reflected in the expert's verbal protocol, hopefully revealing tacit task knowledge. To prevent experts from using their standard "chunks" we use several task manipulations that remove standard knowledge-related features from the task. Collectively, these task manipulations are described as "knowledge-sensitive," because they affect the amount of information available to perform the task, and hence the amount of knowledge the expert can bring to bear on the task. In the visual domains that we have considered, the three main ways of manipulating the task are: 1) area reduction; 2) image quality reduction, and 3) time reduction[5] .

Area Reduction—Area reduction was used by Hochberg (1968) to determine the role of parallel processing in object recognition. We used it to investigate the role of context in object recognition, our second goal in section 2. We applied a series of variable-aperture masks to reduce the area of an image visible at a given time. Each new image was presented to the expert using the smallest-aperture mask. As the expert tried to perform his/her task, the expert indicated the problems caused by the mask: what information, that was usually visible, that he/she needed in order to interpret the image; what comparisons he/she would make, if it were possible. Then, the expert viewed the image again with the next-largest aperture mask, and the knowledge engineer noted what additional information the expert was able to decipher from the image

Image Quality Reduction—Guttman & Webster (1972) and Bloomfield, Wald, & Smith (1979) reduced image quality as a way to provide discriminability metrics for complex visual targets that were embedded in heterogeneous backgrounds. Image quality can be reduced by: interposing neutral density filters; reducing resolution or contrast; defocussing the image; using veiling lights; or increasing viewing distance. We applied this method to the image analysis task by placing a fine grained dot matrix over the image being analyzed. Whichever of these methods of reducing the image quality is used, the general method of acquiring knowledge is the same as with area reduction. Each image is presented first under the poorest image quality conditions, with the expert being asked to perform his/her normal task, and to report the difficulties in doing this. Under the poorest quality conditions, high-frequency information such as internal object detail and

[5] Note that a clustered response paradigm, with the expert reproducing elements of the image, might have been adapted to this application. We did not do this because we were concerned that the response would be influenced by many factors, including image chunks, but also drawing skill, and strategies for maintaining size and distance relationships in the drawing task itself.

texture is the most affected. On subsequent presentations each image may be shown with progressively improved quality, revealing the extent to which high-frequency information is important in interpreting each image.

Time Reduction—Time reduction can be achieved by using a tachistoscopic method of presentation. As with the other methods, the first presentation of the image to the expert is made under the most difficult condition, i.e., with the shortest exposure time. Then, on subsequent presentations, the exposure time is systematically increased. McKeithen et al. (1981) combined a related method with response clustering to examine the structure of computer programs.

4.0 RESULTS OF COGNITIVELY-ORIENTED KNOWLEDGE ACQUISITION APPLIED TO AERIAL IMAGE ANALYSIS

We combined the knowledge sensitive task manipulation described in the previous section, with the goals of a cognitively oriented knowledge acquisition described in section 2, to the investigate the knowledge of expert aerial image analysts. We have piloted the first manipulation with one expert and one novice image analyst, and are planning a test of all three manipulations this year. We developed a preliminary cognitive model based on the information obtained in six hours of knowledge acquisition with these two practicing analysts. We do not claim to have uncovered previously unknown phenomena. Rather, we emphasize that the underlying theory of knowledge and our methodologies led naturally to our findings.

The model we have developed contains a parsed representation of an image, in this case a military air base. Thus, we do not address the problem of image segmentation, nor do we address primarily perceptual questions, such as stereo verification. The knowledge base of the model consists of schemas for representing concepts such as objects and regions, and rules of inference for adding information to the initial representation based on its contents. It is currently capable of recognizing and counting groups of aircraft. We plan to extend the model so that it can deal with the task of evaluating image coherence and change assessment. Figure 1 illustrates some of the schemas and rules used for counting fighter aircraft. Figure 2 illustrates a few of the preliminary interactions with the user, and then the sequence of goals that the model passes through in identifying and counting a group of aircraft. Some of the key features of this preliminary model are presented below, along with some of the informal observations of the knowledge acquisition sessions that justify these features.

4.1 Basic-Level Object Recognition

Like the human analysts, the model reasons about objects initially at a preferred level of abstraction (about categories of aircraft such as fighters and bombers), rather than about superordinate categories (such as vehicles) or subordinate categories (such as MiG-29s). Depending upon the task demands, the level of abstraction (and hence attention to fine detail) can change to a higher or lower one in the representational hierarchy.

Harlow, Trivedi, Conners, & Phillips (1986) noted the inherent arbitrariness of partitioning object features across a hierarchical representation of objects such as the superordinate, basic-level, and subordinate hierarchy we are endorsing. Using the area reduction method, we observed a very rapid object recognition capability for objects such as aircraft and buildings. However, the recognition of objects at a more detailed level (e.g., type of aircraft) was less rapid, with the experts describing a kind of sequential process that is quite compatible with existing theories of object classification (Rosch, 1978). The area reduction method was useful here for identifying features associated with basic-level object recognition and hierarchical class structure. This is because of the importance of global context and relative size information in recognition (see below)—information that was potentially excluded by the reduced view.

With regard to the image quality reduction method, we have only the most informal results,but our impression is that this method will also be useful for identifying relevant features and defining a useful hierarchical class structure in this domain, whereby high frequency information is relevant for discriminating at the subordinate category level (e.g., types of fighters) but low frequency information is relevant for the basic category level (fighters versus bombers).

General Definitions for Model

```
(defschema region
        (is-in-region)
        (is-in-focus no)
        (been-scanned no)
        (borders
                (slot-how-many multiple-values))
        (contains-things)
        (has-target-objects nil)
        (been-counted no)
        (been-tallied no))

(defschema thing
        (is-in-region)
        (been-scanned no)
        (has-adjacent-objects
                (slot-how-many multiple-values)))

(defschema object
        (is-a thing)
        (has-super-type)
        (has-basic-type
        (slot-derivation implicit))
        (has-sub-type
                (slot-derivation implicit))
        (has-shape)
        (has-size)
        (has-parts))
```

Object Definitions for Model

```
(defschema path
        (is-a thru-way)
        (kinds runway road)
        (class-type super))

(defschema runway
        (class-type basic))
```

Knowledge for Object Identification

```
(defrule identify-fighter
        (goal (has-basic-type ?object??))
        (not (has-basic-type ?object?))
        (schema ?object
                (has-shape cylinder)
                has-shape triangle)
                (has-size medium))
        =>
        (assert (schema ?object
                (has-super-type airplane)
                (has-basic-type fighter))))
```

Knowledge for Counting

```
(defrule mark-region-to-search
        ?goal-stack<-(goals $?g)
        ?x<-(current-goal solve-problem)
        (schema ?region
                (instance-of region)
                (been-scanned no))
        =>
        (modify
                (schema ?region
                        (is-in-focus yes)))
        (retract ?goal-stack)
        (retract ?x)
        (assert (goals scan-region-in-focus solve-problem $?g)))

(defrule region-scan-finished
        ?x<-(current-goal scan-region-in-focus)
        (schema ?region
                (instance of  region)
```

020201500-9

Figure 1. Knowledge Base for Image Analysis.

4.2 Context and Relative Size Information

Contrary to at least one perceptual theory for oblique viewing (Biederman, 1987), both our image analysts and our cognitive model use the surrounding context in the object identification task. With both expert and novice photointerpreters, the area-reduction method revealed that the relative size of objects, their immediate surroundings, and overall image context were all important in object identification for aerial image analysis. Context is recognized as important in aerial views because distinguishing features may not be observable. For example, the type of building that appears in an aerial photograph may not be identified on the basis of any particular feature of the building alone, but rather on the basis of the particular features and the relationships of the buildings to such features as access roads and parking lots. Global context, such as position of the building within

the military base, is also important (McKeown, Wilson & McDermott, 1985; Perkins, Laffey, & Nguyen, 1986). However, in contrast to the knowledge acquisition recording methodology provided by McKeown, Harvey & Wixson (1989), we have endeavored to provoke the expert into revealing the content of the contributing context and size information, through the area reduction methodology.

Model: What objects are you interested in?

User: FIGHTER BOMBER

Model: Do you want a regional breakdown or complete tally only?

User: REGIONAL

[Model Processes]:

mark-region-to-search
identify-fighter
scan-for-object-sought-matches
scan-for-object-sought-matches
scan-for-object-sought-matches
scan-for-object-sought-matches
scan-for-object-sought-matches
region-scan-finished
all-scanning-finished
count-objects-by-region
tally-objects
all-counting-finished
report-regional-findings

Model: There are 5 FIGHTER BOMBERS. 5 in region RA.

Figure 2. Sequence of Goals Encountered During Counting Task.

4.3 Domain-Specific Goals and Subgoals

Perhaps the most interesting feature of the preliminary model, compared to standard expert system development, is the influence of human attentional processes on the counting process (Greeno, Riley, & Gelman, 1984). This finding relates more to the goals of our knowledge acquisition than the knowledge acquisition methods we used: we were looking for standard forward-chaining procedures for performing an image analysis task. Like both our expert and novice, the first subgoal is to group the objects in the image by clusters, and then systematically count the objects of each type within each cluster. The groupings (made with grease pencils) help the analyst remember which areas have already been counted and control the transition between one area to another. By contrast, a standard expert system approach would probably make use of a variable for each object type to be identified, and exhaustively scan the image from left to right, incrementing each variable as an instance is encountered. We believe that such subtle processing compatibilities between a user and an automated aid will become increasingly important as the grain size on the interactions between user and aid decrease.

4.4 Searching for a Coherent Representation

Although not yet addressed in our model, Harlow et al. (1986), McKeown et al. (1985), and Perkins et al. (1986) all rely on knowledge about relationships between objects to constrain the interpretation of any individual object. We might say that, driven by the goal of "understanding" the image, experts search for a coherent, internally consistent representation of the image. Thus, internal inconsistencies in the interpretation are cues that the interpretation process has gone wrong. This notion controls the proliferation of multiple hypothetical interpretations about an individual object, groups of objects, or the whole image. While appealing, we find this idea incomplete with regard to both cognitive processes and the true nature of the task.

First, it implies some kind of exhaustive search for coherence, which is surely impractical given the complex and subtle bases on which coherence might be evaluated. We were impressed by the extent of the knowledge underlying consistency and coherence. The process of expanding the viewing aperture in the area reduction manipulation revealed, for example, a required relationship between the aircraft on the base and the oil spots on the runway. A major cognitive issue emerging from this how is the space of possible inferences is controlled during the search for a coherent representation, particularly if these inferences are based on long chains of very general "common-sense" rules.

Second, given the possibility of deception, an image may not be internally consistent. Thus internal inconsistencies are interpretable and significant to the image analyst; they will be investigated rather than discarded as misinterpretations. Here, the issue is the knowledge that controls the computation required to initiate, pursue, and confirm or reject an hypothesis of deception. It seems to us that experts do not test every possible inconsistency for deception. Rather, they first test whether or not a few of the obvious inconsistencies can be explained easily as a consequence of normal activity. The analyst develops an hypothesis of this normal activity and searches the image for a few items of specific evidence that challenge their hypothesis (Wason & Johnson-Laird, 1972), halting the process when a hypothesis is obtained for the image that is supported by the identified features.

5.0 CONCLUSIONS

In our future work we plan to systematically test our approach to knowledge acquisition compared to more standard interviewing techniques, in both aerial image analysis, and pilot situation awareness during short-term planning tasks. We have repeatedly noted the risk of departures from the veridical task environment during knowledge acquisition. Certainly, knowledge-sensitive task manipulation carries the same risk. The aerial image analysis test will build on the work presented here. The pilot reasoning analysis will make use of the simulation environment we have developed for another project. Using a touch sensitive menu in the cockpit of this environment, the pilot can request information as needed. This will provide an observable and permanent record of his or her information needs. Moreover, within this simulation environment, we can systematically control the automatic presentation of pilot information requirements in concert with his or her performance of piloting tasks. By limiting the amount of information available at any time, we hope to force pilots to explicitly express their information requirements, thereby providing a basis for inquiring about the underlying reasoning that supports situation awareness. Hall & Phelps (1983) describe a similar method for cognitive analyses of information acquisition in decision-making, although an expert system application is not explicitly considered.

Our intention is to encourage the integration of the techniques and goals of cognitive modeling into the mainstream of knowledge acquisition and representation. In this paper, we have described some of the standard problems associated with knowledge acquisition. We distinguished between the dependent measures used for knowledge acquisition and the knowledge acquisition task from

the expert's perspective, as background for our approach to the problem of identifying tacit knowledge based on knowledge-sensitive task manipulation. We have illustrated some of the immediate benefits of such an approach in terms of the insights one might gain about the nature of expertise and the knowledge required for expert level performance. The primary benefit of our approach to knowledge acquisition may ultimately be in facilitating the necessary interactions between an expert system and the human user in complex domains—by maximizing the similarity of problem representations—and hence reasoning processes—between the system and the user.

REFERENCES

Anderson, J. R. (1983). The Architecture of Cognition. Cambridge, MA: Harvard University Press.

Anderson, R. C., and Ortony, A. (1975). On putting apples into bottles—a problem of polysemy. Cognitive Psychology, 7, 167-180.

Berry, D. (1987). The problem of implicit knowledge. Expert Systems, 4(3), 144-151.

Biederman, I. (1987). Recognition by components: A theory of human image understanding. Psychological Review , 94(2), 115-147.

Bloomfield, J. R., Peio, K. J., Lehman, E. F., Masters, R. M., and Boettcher, K. L. (1989). Flight decision characterization. Final Report on Task 4 of CAT-ALYST Program under contract no. F33615-88-D-0533 for Air Force Systems Command, OL-AC MSD/YA (CAT), Wright-Patterson Air Force Base, OH.

Bloomfield, J. R., Wald, J., and Smith, D. A. (1979). Final report on the optimal placement of military targets in cluttered terrains. Honeywell Technical Report for U.S. Army Mobility Equipment Research and Development Command, contract no. DAAK70-77-C-0056.

Bloomfield, J. R., and Shalin, V. L. (1989). "Knowledge Acquisition Techniques: Diagnosis and Prognosis," in Proceedings of the 1989 Ergonomics Society Annual Conference, England.

Boose, J. H. and Bradshaw, J. M. (1987). Expertise Transfer & Complex Problems: Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems. Special Issue of the International Journal of Man Machine Studies on the AAAI knowledge Acquisition for Knowledge Based Systems Workshop.

Chase, W. G., and Simon, H. (1973). Perception in chess. Cognitive Psychology, 4, 55-81.

Chi, M. T. H., Feltovich, P., and Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. Cognitive Sciences, 5, 121-152.

Chi, M. T. H., Glaser, R., and Rees, E. (1982). Expertise in problem solving. In R. J. Sternberg (Ed.), Advances in the Psychology of Intelligence. Hillsdale, NJ: Erlbaum.

Collins, H. M., Green, R. H., and Draper, R. C. (1985). Where's the expertise? expert systems as a medium of knowledge transfer. In M. Merry (Ed.), Expert Systems '85, Cambridge University Press.

Davis, R. (1979). Interactive transfer of expertise: Acquisition of new inference rules, Artificial Intelligence, 12, 121-157.

Duncker, K. (1945). On problem solving. Psychological Monographs, 58(5), Whole No. 270.

Edman, T. R., Cochran, E. L., and Hadden, G. (1988). KLAMSHELL, Scientific Honeyweller 9, 6-11.

Ericsson, K., and Simon, H. A. (1986). Protocol Analysis: Verbal Reports as Data. Cambridge, MA: MIT Press).

Fitts, P. M. (1964). Perceptual-motor skill learning. In A. W. Melton (Ed.), Categories of Human Learning. New York: Academic Press.

Greeno, J. G. (1977). Process of understanding in problem solving. In N. J. Castellan, D. B. Pisoni, and G. R. Potts, Cognitive Theory (Vol. 2), Hillsdale, NJ: Erlbaum.

Greeno, J. G., Magone, M. E., and Chaiklin, S. (1979). Theory of constructions and set in problem solving. Memory and Cognition, 7, 445-461.

Greeno, J. G., Riley, M. S., and Gelman, R. (1984). Conceptual competence in children's counting. Cognitive Psychology, 16, 94-143.

Greeno, J. G., and Simon, H. A. (1986). Problem solving and reasoning. In (Eds.) R. C. Atkinson, R. J. Hernstein, G. Lindsey, and R. D. Luce, Steven's Handbook of Experimental Psychology. New York, NY: Wiley.

Guttman, H. E., and Webster, R. G. (1972). Determining the detectability range of camouflaged targets. Human Factors, 14, 217-226.

Hall, J. and Phelps, R.H. (1983). Non laboratory techniques for the study of cognitive and decision making processes: A description and selected bibliography. U.S. Army Research Institute for the Behavioral and Social Sciences, Research Note 83-45.

Harlow, C. A., Trivedi, M. M., Conners, R. W. and Phillips, D. (1986). Scene analysis of high resolution aerial scenes. Optical Engineering, 25(3), 347-355.

Hochberg, J. (1968). In the mind's eye. In Contemporary Theory and Research in Visual Perception, ed. R. N. Haber, NY: Holt, Rinehart, Winston.

Hoffman, R. R. (1987). The problem of extracting the knowledge of experts from the perspective of experimental psychology, AI Magazine, 5, 53-67.

Hoffman, R. R. (1984). Methodological preliminaries to the development of an expert system for aerial photo interpretation. U.S. Army Engineer Topographic Laboratories technical report number ETL-0342.

Howarth, C. I. (1988). The relationship between objective risk, subjective risk, and behavior. Ergonomics, 31, 527-535.

Hyde, T.S. and Jenkins, J.J. (1969). The differential effects of incidental tasks on the organization of recall of a list of highly associated words. Journal of Experimental Psychology, 82, 472-481.

Klein, G. A. (in press). Recognition-Primed Decisions. In Rouse, W. R. (Ed.), Advances in Man-Machine Systems Research, Vol. 5.

Kowalski, B. & VanLehn, K. (1988). Inducing subject models from protocol data (Tech. Rep. PCG-15). Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA.

Laird, J. E., Rosenbloom, P. S., and Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism, Machine Learning, 1, 11-46.

Langley, P. and Ohlsson, S. (1984). Automated Cognitive Modeling. In Proceedings of the National Conference on Artificial Intelligence, William Kaufmann, Inc., Los Altos.

Larkin, J. H., McDermott, J., Simon, D. P., and Simon, H. A. (1981). Expert and novice performance in solving physics problems. Science, 208, 1335-1342.

Larkin, J. H., McDermott, J., Simon, D. P., and Simon, H. A. (1979). Models of competence in solving physics problems. Carnegie-Mellon University CIP Working Paper Number 408.

Levi, K., Shalin, V., and Perschbacher, D. (1988). "Automating Acquisition of Plans for an Intelligent Assistant by Observing User Behavior," presented at the Third Workshop on Knowledge Acquisition for Knowledge-Based Systems.

Lewicki, P. (1986). Processing information about covariations that cannot be articulated. Journal of Experimental Psychology: Learning, Memory and Cognition, 12(1), 135-146.

McKeithen, K. B., Reitman, J. R., Ruter, H. H., and Hirtle, S. E. (1981). Knowledge organization ad skill differences in computer programmers. Cognitive Psychology, 13, 307-325.

McKeown, D.M., Harvey, W.A. and Wixson, L.E. (1989). Automating knowledge acquisition for aerial image interpretation. Computer Vision, Graphics, and Image Processing, 46, 37-81.

McKeown, D. M., Wilson, H. A., and McDermott, J. (1985). Rule-based interpretation of aerial imagery. IEEE Transactions on Pattern Analysis, 7(5).

Mitchell, T. M., Mahadevan, S., and Steinberg, L. I. (1985). LEAP: A learning apprentice system for VLSI design, LCSR Technical Report 64, Rutgers University, Dept. of Computer Science, New Brunswick, NJ.

Moninger, W., and Stewart, T. (1987). A proposed study of human information processing in weather forecasting. Cited by Hoffman, R. R., 1987, The problem of extracting the knowledge of experts from the perspective of experimental psychology, AI Magazine, 8, 53-67.

Newell, A., and Simon, H. (1972). Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall.

Nisbett, M. J., and Wilson, T. D. (1977). Telling more than we can know: Verbal reports on mental processes. Psychological Review, 84(3), 231-259.

Nissen, M. J., and Bullemer, P. T. (1987). Attentional requirements of learning; evidence from performance measures. Cognitive Psychology, 19, 1-32.

Olson, J. S., and Rueter, H. H. (1987). Extracting expertise from experts: Methods for knowledge acquisition. Expert Systems, 4(3), 152-167.

Osgood, C. E. (1953). Method and Theory in Experimental Psychology, New York: Oxford University Press.

Palmer, S. (1975). Visual perception and world knowledge: Notes on a model of sensory and cognitive interaction. In D. A. Norman, D. E. Rumelhart and the LNR Research Group, Explorations in Cognition, San Francisco: W.H. Freeman & Co.

Perkins, W. A, Laffey, T. J., and Nguyen, T. A. (1986). Rule-based interpreting of aerial photographs using the Lockheed Expert System. Optical Engineering, 25(3), 356-362.

Reitman, J. S. (1976). Skilled perception in GO: Deducing memory structures from inter-response times. Cognitive Psychology, 8, 336-356.

Reitman, W. (1965). Cognition and Thought: An Information Processing Approach, New York, Wiley.

Rosch, E. H. (1978). Principles of categorization. In E. Rosch and B. B. Lloyd (Eds.), Cognition and Categorization, Hillsdale, NJ: Erlbaum.

Schneider, W. (1985). Toward a model of attention and the development of automatic processing. In M. I. Posner and O. M. Marin (Eds.), Attention and Performance XI. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Shalin, V. L., and Boy, G. A. (in preparation). Report on IJCAI-89 Workshop on Integrated Human-Machine Intelligence in Aerospace Systems. To appear in AI Magazine, Menlo Park, CA, AAAI.

Shalin, V. L., Levi, K. R., and Perschbacher, D. L. (1989). Identifying Knowledge Base Deficiencies by Observing User Behavior. To be presented at the Sixth International Workshop on Machine Learning, Ithaca, NY.

Shalin, V. L., Perschbacher, D. L., and Jamar, P. G. (1988). "Intent Recognition: An Emerging Technology," Proceedings of the 2nd International Conference on Man-Machine Interface and Artificial Intelligence in Aerospace.

Shalin, V. L., Bloomfield, J. R., Bullemer, P. J., Bhanu, B., and Shelton, C. (1988). "Cognitive Modeling-Based Knowledge Acquisition for Photo Interpretation," presented at the Intelligence Community's Sixth Annual AI Symposium.

Shalin, V. L., Wisniewski, E. J., Levi, K. R., and Scott, P. D. A formal analysis of machine learning systems for knowledge acquisition. International Journal of Man Machine Studies, 29, 429-446.

Shiffrin, R. M., and Schneider, W. (1977). Controlled and automatic human information processing (II): Perceptual, learning, automatic attending, and a general theory. Psychological Review, 8, 127-190.

Simon, H. A. (1969). The Sciences of the Artificial. Compton Lectures, Cambridge, MA: MIT Press.

Simon, H. A. (1973). The structure of ill-structured problems. Artificial Intelligence, 4, 181-201.

Tulving, E., and Thompson, D. M. (1973). Encoding specificity and retrieval processes in episodic memory. Artificial Intelligence, 28, 235-240.

VanLehn, K. (1983). On the representation of procedures in repair theory. In. H. P. Ginsburg (Ed.), The Development of Mathematical Thinking. New York: Academic Press.

VanLehn, K. and Garlick, S. (1987). Cirrus: an automated protocol analysis tool. In Langley, P. (Ed.) Proceedings of the Fourth Machine Learning Workshop, Los Altos, CA: Morgan Kaufmann.

Wason, P. C., Johnson-Laird, P.N. (1972). Psychology of Reasoning, Cambridge, MA: Harvard University Press.

Waterman, D. A. (1986). A Guide to Expert Systems. Reading, MA: Addison Wesley.

Willingham, D. B., Nissen, M. J., and Bullemer, P. T. (in press). On the development of procedural knowledge. Journal of Experimental Psychology: Learning, Memory and Cognition.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Knowledge Acquisition for Advice Systems : FUSION, A Network Cognitive Support System

Maurice Sharp
Knowledge Sciences Institute
University of Calgary
Calgary, Alberta, Canada T2N 1N4
Email : sharp@ksi.cpsc.UCalgary.CA

ABSTRACT

This paper analyzes requirements and techniques for knowledge acquisition to support advice to users of large-scale heterogeneous networks. The complex dynamics of such networks requires embedded knowledge acquisition as an integral part of user support and advice systems. Some aspects of the knowledge acquisition required fall into conventional information techniques such as computer-assisted software engineering and others relate to developments in knowledge acquisition for knowledge-based systems. This paper gives and integrative framework for user support in large-scale heterogeneous networks that encompasses the main aspects of the problem.

A computer, or network of computers, can be seen as a set of resources in the form of hardware and software. Users approach computers with intentions that can be fulfilled by some combination of tasks. Even within a single workstation environment, users may experience difficulty in specifying tasks that satisfy their intentions, and in matching tasks to resources. As the scale of heterogeneous networks increases, the level of difficulty is likely to increase since there is no single standard for communication at the application level.

One solution to the problem is to support the user in specifying tasks and mapping them to the available resources. This requires knowledge of both the resources on the heterogeneous network, and a characterization of the user. By viewing the exchange between the user and the system as a layered communication protocol, it is possible to structure the two components within a single model.

The internal system view contains knowledge about the network, the software, hardware, topology, and loading. The system view of the user contains knowledge of the users intentions, tasks, preferences, and knowledge of the system.

Traditional knowledge acquisition techniques can be used for some of the needed knowledge. However, some sources require the extension of existing technologies to enable their acquisition.

Introduction

The current trend in computer systems is towards large scale heterogeneous networks combining differing architectures (Gaines 88b). To facilitate the sharing of information and computing power within these heterogeneous networks, standards for inter–machine communications have been developed such as MAP, TOP, and the International Standards Organization open system interconnect (ISO/OSI). The key to all of these standards is the definition of a virtual machine (or machines) that allows each element of a heterogeneous network to be treated as homogeneous.

The sharing of information and compute power will provide users with access to more services than current standalone systems and small scale networks. However, users are already having difficulty accessing services in small scale environments since they are required to be acquainted with both the location of the service, and how to interact with the machine that offers it. As the scale of networks increases, the amount of knowledge that the user needs increases.

One approach is to provide the user with the knowledge needed. The method used should minimize the amount of work the user needs to do. This suggests a knowledge based advice and management system.

Part of the knowledge required already exists in the heterogeneous system. The difficulty lies in acquiring this system knowledge. Current knowledge acquisition technologies are based on expertise transfer or machine learning. Both methods can be used to gain knowledge of the user, particularly since the user may be viewed as, potentially, an 'expert' on themselves.

Both methods are also applicable to acquiring some aspects of the knowledge contained in the heterogeneous system. However, there are also more direct routes to system information that may be more accurate, timely and reliable than expertise transfer or machine learning. These direct routes do not involve some of the more difficult aspects of knowledge acquisition. But using them effectively leads to problems of knowledge acquisition that are not usually considered but are important in their own right in an integrated system.

The overall goal of the system, called FUSION, is to aid the user in the completion of tasks in a heterogeneous network environment. This is accomplished in three main ways :

1 - Provide a structured and consistent interface to the advice system
2 - Provide a representation of the current system state
3 - Provide task completion advice based on user intentionality

Knowledge plays a critical role in FUSION since it is required for items two and three. This paper gives an overview of the sources of knowledge and some of the acquisition methods used. The design of FUSION is used as a background for the discussion.

Before commencing, there are a number of terms which should be defined. A *heterogeneous network* is one that is composed of machines of possibly differing architectures, configurations, and operating systems. A *large scale network* spans a corporate or academic entity, and may cover a large geographic area. The *resources* of the system refer to the programs, data, and processing power; an example would be a mail system, or a word processor.

FUSION

A user comes to a computer with one or more intentions that can be satisfied by completing a number of tasks (Norman 84). A user also has a conceptual (cognitive) model of the system and its resources (Smith et al 83). The difficulty lies in first specifying the tasks, and then matching the tasks to the resources. The user may already be uncertain about how to accomplish this on a standalone computer.

This uncertainty is likely to increase as the scale of available resources increases. In a large scale network there are many resources, some of which duplicate, or closely duplicate, the functionality of others. Similar resources may be accessed by very different invocations, some may be heavily loaded and others lightly loaded, some expensive and fast, others cheap but slow. These complications make it difficult for the user to form a consistent model of the system. This is the major problem in the effective utilization of large scale heterogeneous networks.

There are two separate parts to the problem, a system with a set of resources, and a user trying to access these resources to accomplish tasks. The system can be seen as a virtual interface to a distributed network. This idea of *virtuality* can be applied to the systems view of the user, giving a model of two entities (computer and/or human) communicating over virtual channels (Taylor 88). The reason for human/computer communication is a user accessing a set of resources to accomplish tasks. Since the two entities are a computer and a human, FUSION needs to contain a model of the heterogeneous network and of the user.

The computer can assist the user if the system is designed to include and use knowledge. At the highest level of organization, knowledge of the user and the environment is required. The choice of how to break down this high level will effect the ease of modeling the system and the user.

The breakdown of knowledge defines the main tasks of FUSION. For the system side of the model it is state representation, and for the user side it is advice giving. This suggests two interacting parts, one that models the heterogeneous network (*Internal System Model*), and one that represents FUSION's model of the user (*System View of User*). This simple design is shown below :

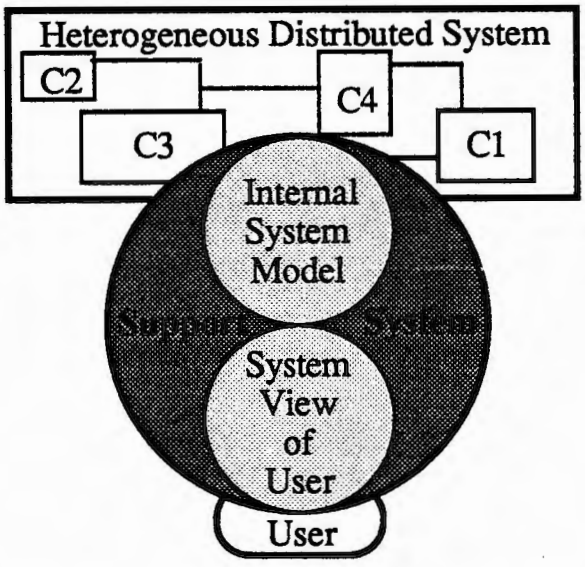


Figure 1 : Basic Model of FUSION

This model is still at too coarse a level for isolating types of knowledge. In her paper in intelligent user interfaces, Rissland (84) gives seven sources of knowledge that can be used as a basis :

Source	Elements
User	expertise, style, preferences, history
User's tasks	context, purpose, ultimate use of results
Tools	protocols, parameters, side-effects, resources, costs
Domain	epistemological, measures, concepts
Modalities	display capabilities and methods, interaction devices
How to interact	intervention, adaptation, modes, user behaviour
Evaluation	effectiveness of interaction

Table 1 : Rissland's Sources of Knowledge

In terms of the basic architecture for FUSION, the internal system model contains the knowledge of tools, modalities, and how to interact. The system

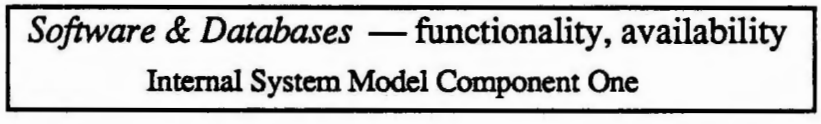
view of the user contains the knowledge of the user and user's tasks. Knowledge of how to interact is built into the support system itself, as is evaluation knowledge. There is currently no provision for specific domain knowledge in the system. To see where these sources fit in requires looking at each side of the FUSION model.

FUSION - Internal System Model

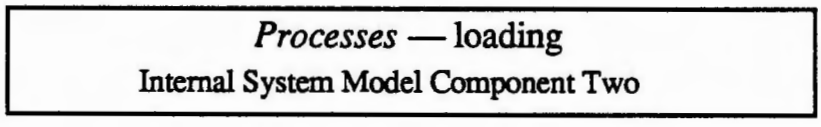
This part of FUSION is responsible for maintaining a representation of the current state of the network that can be used to aid in completing user tasks. A user task is a step in the fulfilment of an intention. This can range from sending mail, to editing a document, to copying one line, to typing one character (Norman 84).

A task can be viewed as the execution of a plan, or in computer terms, an algorithm. This algorithm may need a software package, several software packages, or part of a package. Thus from an internal system point of view, task fulfilment is related to the functionality of the software in the system. It is also important to know what software is currently available. This affects what elements are available to construct a plan from. In addition, the functionality of the interface needs to be known. This will allow the system to aid the user in interacting with the software.

Software is procedural in nature. It specifies a set of steps that needs to be accomplished. Humans also have declarative knowledge. The analogous entities in a network are databases. These two sources give the first part of the internal system model :



In a user, procedural and declarative knowledge are static quantities. To be of use they must be instantiated in some form. The same is true for computers. For the software to actually accomplish a task, it must be instantiated in the form of a process. The distribution of these processes (or *loading*) gives an indication of how busy elements of the network are. If the users current task is computationally intensive, knowledge of loading will allow the task to be distributed in a reasonable fashion. In terms of Rissland's (84) knowledge sources, tools includes those available and being used. This is the second source of knowledge :



At the lowest level, processes are a set of instructions executing on some processor (or processors). A particular piece of software may only run on specific architectures, a package may require advanced graphical functions not available on low end terminals. Thus a knowledge of the type of machines (*platforms*) available is needed. Rissland (84) refers to this as knowledge of modalities.

It is also useful to know where particular platforms are. If the software with the required functionality needs a particular type of platform, the system needs to know where this platform exists. The layout will also help to reduce communication costs between elements. Thus a knowledge of the actual layout (*topology*) of the network is required.

Hardware — network topology, platforms
Internal System Model Component Three

The three sources of knowledge allow the internal system model to produce a complete model of the state of the system.

FUSION - System View of User

The main goal of the system is to aid a user in task completion. This is done in two main ways, support of the users cognitive model (*cognitive support*), and an advice paradigm. The cognitive model is supported at the user interaction level through the presentation of the resources. This uses a well designed and consistent interface, but does not require knowledge of a specific user to be effective (Rissland 84, Greenberg 88).

Literature on advice falls into two categories, expert systems and tutoring systems. The difference between the two types is the initiator of the advice. In an expert system, advice is given in answer to specific queries, and is thus user initiated. A tutoring system may give this type of advice, or the system may take the initiative.

As mentioned above, users are not experts when they com to the system, nor are they at a consistent level of expertise with all components. This suggests that FUSION be able to initiate advice, as well as answer questions. Adding an intelligent tutor will increase the teaching ability of the advice system.

The advice paradigm does require knowledge of the user on several levels. At the highest level, the advice should be relevant to the user. It is pointless to give a message about memory exceptions and core dumps when a user is attempting to mail a letter. Meaningful advice requires knowledge of the users intentions (Norman 84, Carroll and McKendree 87). This forms the first part of the system user view :

Intentions — goals, objectives
System User View Component One

After forming an intention, a user needs to fulfil it. This requires a planning process which draws on the users knowledge of the system, as well as their strategic knowledge on how to plan. In the actual planning action, the user will be influenced by their knowledge of the world. Thus the systems user view needs access to all three types of knowledge forming the second part :

Knowledge — world, planning, system
System User View Component Two

After the intention has been broken down into general tasks, and the tasks have been matched to resources, the user will need to interact with each resource. This interaction may differ from resource to resource. The user may also be at different levels with different resources. Thus some knowledge of a user role specific to a resource is needed. In addition, users tend to reuse sets of resources in particular ways (Greenberg 88). This knowledge of user role forms the final part of the system user view :

User Role — tasks, preferences
System User View Component Three

These three components give a model of the user that are used to assist users in completing intentions. Combining these three components with those from the internal system model, and the basic architecture yields a more complete picture of the system :

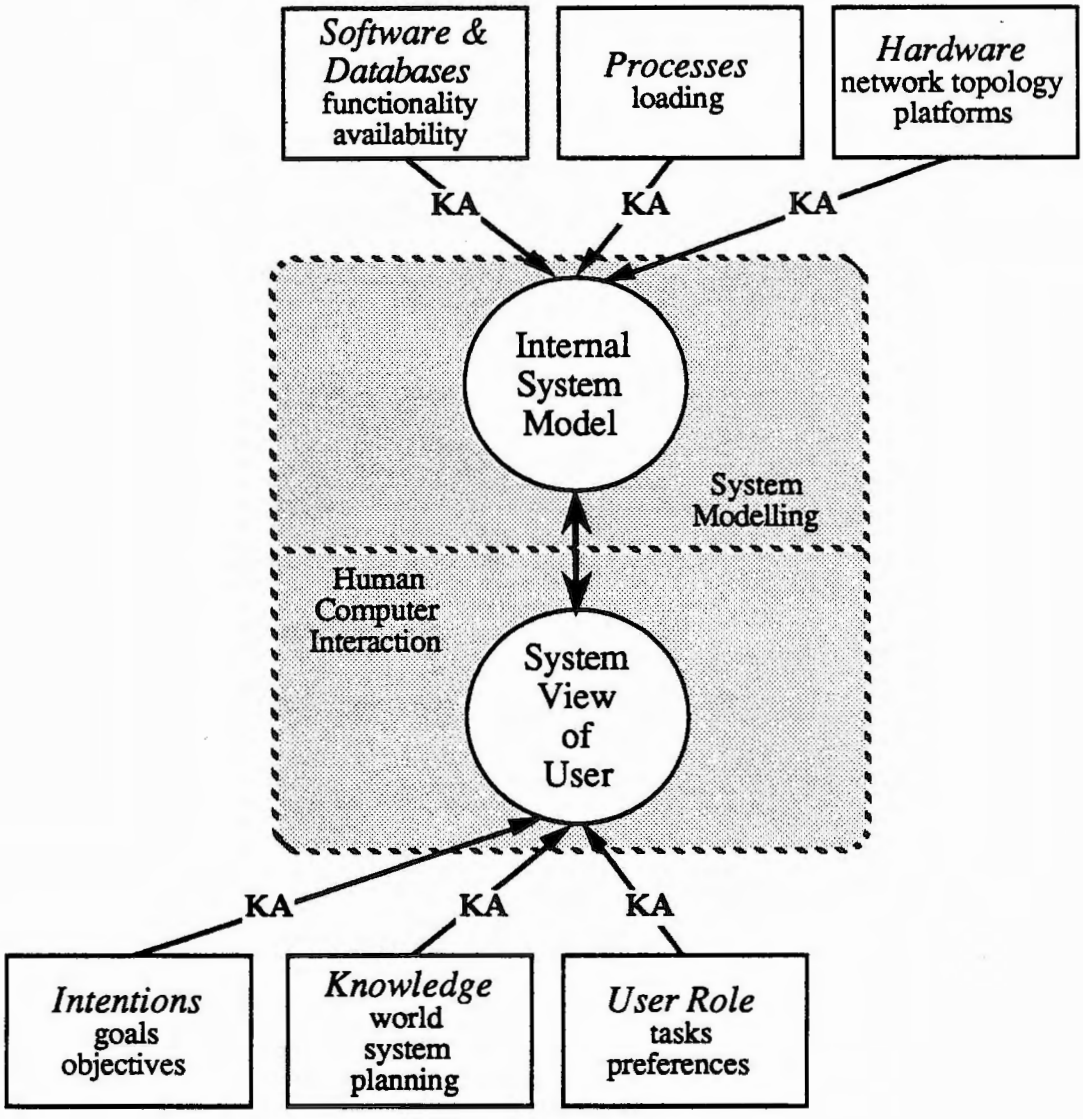


Figure 2 : The FUSION System

Knowledge Sources and Acquisition in FUSION

The components of the expanded view correspond to major types of knowledge needed about both sides of the communication between user and heterogeneous system. The system side contains information on the available resources :

Software & Databases provide the functionality for completion of user tasks. This includes the functionality of the user interface. Knowledge of where applications and databases reside is also needed. These are passive entities, they can do no work themselves.

Processes are the instantiation of software and databases. These are the active entities on the network. Knowledge of how busy machines are (the loading) is important. It can be used to determine where best to run a particular process.

Hardware has two aspects. The topology (layout) of the network provides information about how to access software and databases. It also figures into time costs. Knowledge of platforms (machine architectures) can be used in determining what type of interface to present to a user. A text terminal would not be able to support general graphics.

The user side has information about the individual user :

Intentions represent the goals and objectives of the user. These may change over time, and there may be sub–levels. For instance, a user may start by writing a letter to a friend, but digress into reading mail.

The *Knowledge* of the user ranges from the very general to the specific. The world knowledge affects the way the user thinks about the system. Knowledge of the system allows a user to plan the completion of an intention. Planning is the strategic knowledge about how to plan. All of this knowledge can change over time.

User Roles represent a user trying to fulfil a specific goal. This may be editing a document, reading mail, or building an application. Each role has a set of preferences and tasks associated with it. The preferences are analogous to a workbench, and the tasks are the tools that can be used (Greenberg 88).

The rest of this section will discuss the sources of the knowledge in each of the components. It will also discuss some of the acquisition techniques used.

Software & Databases

This knowledge is mainly used to match user tasks to available resources. It could also be used to explain why a resource no longer works when it is upgraded. Information about where to find an old version is included in this knowledge. This information can be maintained by use of versioning tools.

The functionality of the resource is needed for task matching. The knowledge of functionality resides in three places, the manual, experienced users heads, and the design specification.

The current role of text analysis in knowledge acquisition is either to provide a knowledge engineer with a basic view of the domain (Linster 88), or to provide a list of useful concepts for an automatic acquisition process (Woodward 88). Neither of these methods will give specific information on functionality. A solution is to assign the task of acquiring the functionality knowledge in the manual to a human. This solution can be expensive, especially since human time tends to be more valuable than computer time.

Automatic acquisition from an expert is possible, but is also not a good solution. The time of an expert is valuable. This is important since any time the resource is updated, the knowledge will need to be updated. It also may not be the same expert giving the knowledge.

A better source is the design specification. To use this source requires making the knowledge of functionality explicit. This is done by using computer aided software engineering (CASE) tools during the design of the resource. The process of acquiring knowledge about functionality now becomes one of using the knowledge from CASE tools. This is reasonable for large software packages, since the industry tends to use these tools.

Processes and Hardware

Though these two components have different levels of knowledge, they are both covered by network management techniques. Pasquale (88) gives a set of characteristics that an expert distributed resource manager should exhibit : ability to deal with uncertainty, ability to react to changing situations, interpretive and inferential capabilities, ability to collaborate and coordinate, and high performance. These criteria can be reduced since some are equivalent. The ability to deal with uncertainty stems from the interpretive and inferential capabilities. The ability to react is related to high performance.

Knowledge based systems fit the requirement for dealing with uncertainty, and distributed problem solving is also possible (Durfee 88). The problem will be the performance time. This requirement stems from management needs at a very low level of network operations. Such elements as packet routing and network error detection/correction, are not as critical to user support. Thus the time factor is not as important in FUSION.

At the hardware level there are two types of knowledge required. The topology information is generally available in configuration files, such as those found in TCP/IP. System administrators are the source of this knowledge.

The knowledge of platforms is functional and covers two levels. One is the nature of the devices present. This is information the characteristics of input and output devices, such as the type of screen, and the presence of a mouse.

Most systems contain ways to access this information. A simple example is the TERMCAP database in UNIX systems.

The other level of functionality is the communication protocol. There is a definite move toward standardization of some levels of this interface (for example ISO/OSI). The major difference occurs at the operating system level. There is a great deal of difference between IBM-JCL and UNIX. The simplest way is to have information on the type of operating system present in the topographical knowledge.

Automatic acquisition of this knowledge can be done in a simplistic way. Try to talk with the particular platform using all the dialects that are known until success is achieved, or all the dialects are exhausted. This is an exhaustive search, and not very efficient. It is possible to place constraints on the search by examining the hardware of the platform. This should give some clues as to possible operating systems.

Intentions

The idea of acquiring user intentions seems to be very difficult. However, the problem can be heavily constrained. The only intentions of interest are those relevant to task completion. The fact that the user wants to mail Fred Blogs is important, the fact that he wants to convince Fred to go fishing for the weekend is not directly relevant.

The first level of constraint can take place in the interface design. It should be structured to acquire general intentions by initial user actions. An example would be presenting the user with the screen below :

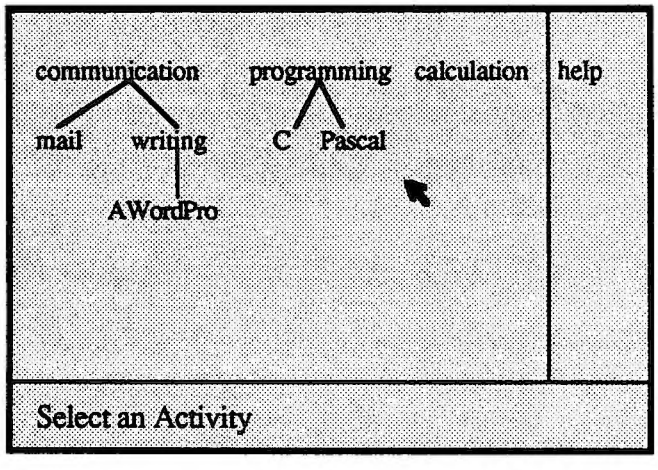


Figure 3 : A Screen for High Level Intentions

In this example, the user is required to specify what general activity they wished to perform before proceeding. This can be as general as the intention

to mail, but it could also be more specific. Each node in the hierarchy of intentions contains information on what the intention requires. Since the user can add nodes, they could create a node called 'mail fred' that contained all the resources used for that operation.

The next level of intentions can be acquired by observing that FUSION is a cooperative advisory system. Kass & Finin (88a, 88b) present a set of user model acquisition rules that can be used implicitly to build a model of the user sufficient for tailoring explanations. This is sufficient for helping novice users gain familiarity and for aiding advanced users in their tasks (Kass & Finin 88a).

Another important observation is about the length of time the knowledge is needed. Since a user comes to a computer with an intention, the knowledge associated with it need only exist for the duration of the relevant session. When intention *A* is completed, the knowledge for it is no longer needed. Thus, unless specifically requested, the system need not remember a large amount of data. In concrete terms, an intention *A*, corresponds to a temporary node *A* in the hierarchy of figure 9. This node is a workspace (Greenberg 88) associated with intention *A*. Unless the user wishes to keep the workspace, the system can dispense with the node once the action is completed.

Thus intentions can be ascertained in two ways. The high level intentions can be found by constraining the user to specify them. Finer details can be gained using both interview and observation.

User Knowledge

According to Norman (84), a user cycles through four stages of activities, forming and intention (intention), selecting an action (selection), executing the action (execution), and evaluating the income (evaluation). The user intentions component in FUSION contains high level intentions, the user knowledge component is used to break this high level down into actions. This corresponds to Norman's selection stage :

> "Support for the selection stage comes principally from memory aids ... that allow the user to determine the possible commands and their modes of operation, prerequisites and implications. Selections can be enhanced by "workbenches" that collect together relevant files and software support into one convenient location." (p. 368)

FUSION uses memory aids in two ways. The interface is set up as a hierarchical graph of activities (high level intentions), and the advice system can give information on commands and resources. Norman also suggests four sources for knowledge of commands :

1 - Retrieval from memory
2 - Reminded by another person, the system, or a manual
3 - Construct or derive possibilities
4 - Teaching by another person, the system, or a manual

For memory retrieval, the user already knows what commands are needed. Source 2 is an example of the advisory system, with source 4 adding a tutorial element to this. Source 3 is a candidate for knowledge acquisition, since it involves the users ability to plan, and their knowledge of the system.

This type of activity is a synthesis task, and specifically a planning task. It is possible to use an explanation based learning approach to knowledge acquisition to assist the user in planning to satisfy the intentions (Levi et. al. 88). In this case, the planner would not be autonomous, instead it would interact with the user in an interview style to refine the plan. This is also close to a tutorial style, thus as the plan is constructed, the user learns more about the resources of the system and how they can be combined.

User Role

A user role refers to the set of characteristics and preferences associated with a user performing a particular task. The characteristics may differ between applications, and may not reflect the level of sophistication of the user.

> "It also inclines more experienced users to stagnate, for when situations occur that could be handled by new procedures, these users are likely to stick with the procedures they already know, regardless of there efficacy." (Carroll and McKendree 87, p. 14)

This indicates that a knowledge of how a user interacts with a resource is useful in two ways. First, the user is likely to be consistent in the way they perform particular operations. The system can then suggest and perform actions sequences. Secondly, it gives a basis from which the system can tutor the user on other functionality provided by the resource.

This type of knowledge is performance based. It consists of sequences of activities, and can be acquired by observing user actions. Since FUSION is designed to aid at the resource usage level, it is not concerned with interaction inside the resource. Thus it does not need to know about sequences of commands in a text editor, mearly that editor *Y* is used to fulfil task *X*.

The system should also be able to present a user with an environment that is suitable to the current task. It should acquire knowledge of window setups, command aliases, and preferred applications. This knowledge can also be acquired by observing user actions.

Some Technologies for Knowledge Acquisition in FUSION

To satisfy the goal of aiding a user in the completions of tasks in a large scale heterogeneous network requires a diverse range of knowledge. As such, the knowledge acquisition performed by the system is also varied. Some methods fall outside the current literature on knowledge acquisition, but do exist in other literature. Thus the system needed to satisfy the goal must integrate different technologies. The FUSION system sets out to do this.

Table 2 gives the sources of knowledge used and some of the techniques used to acquire the knowledge.

Source	Technique(s)
Software & Databases functionality availability	versioning tools CASE
Processes loading	scripts / stats reporting daemons
Hardware network topology platforms	reporting daemons
Intentions goals objectives	interface structure cooperative advisory dialog rules
Knowledge world system planning	interact with user in a problem-solving mode
User Role tasks preferences	interact with user using interview for refinement observe user behaviour

Table 2 : Techniques of Acquisition in FUSION

Current knowledge acquisition methods cover most of the system view of the user. Acquiring intentions is outside this scope, but is covered above. Since part of the aim of this paper is to address knowledge acquisition techniques that are outside the norm, the rest of this section will concentrate on the internal system model.

Most of the techniques for the internal system model are not strictly recognized as knowledge acquisition. However, they do provide solutions to some problems that are recognized as requiring knowledge based solutions. These include (taken from Boose 88, page 3–1):

Analysis tasks
    Diagnosis - inferring system malfunctions from observables.

Synthesis tasks
    Configuration - configuring collections of objects under constraints in relatively small search spaces
    Design - configuring collections of objects under constraints in relatively large search spaces.

Tasks combining analysis and synthesis
    Monitoring - comparing observables to expected outcomes.

Diagnosis requires the acquisition of the observables. Configuration and design require acquisition of the collections of objects, and of the constraints. Monitoring requires acquisition of the observables. The observables in FUSION are the hardware platforms, process distributions, and network topology. Software and databases are the collections of objects to be constrained by both their functionality and the user's needs.

Software & Databases : CASE

There are two pieces of knowledge required about the software and databases, what they do (*functionality*) and where they are (*availability*). Both types of knowledge can exist in computer aided software engineering (CASE) environments.

The paper by Houghton (87) surveys both the characteristics of tools and the state of the art in CASE. Part of the software life cycle is to define its functionality. Using a tool to specify the functionality makes the knowledge explicit. All that is required is for FUSION to acquire the knowledge from the CASE tools representation.

Another phase of the life cycle is operations and maintenance. During this phase, tools are used to track changes functionality. These tools also track configuration releases. This knowledge is useful when a user encounters a problem with a new release of a package. An example is code that compiled under the old version but not under the new. The configuration and release knowledge (versioning) tells the system that a new compiler is in use.

This versioning capability acquires only part of the knowledge of availability. In addition to knowing what the current version is, FUSION must know where to locate the software (or where old version are). A system

administrator could inform FUSION when new software is installed or removed. However this is both tedious for the administrator and error prone.

A better solution is to have FUSION automatically check for new software. At the very least, the modification dates of executables can be checked. If they have changed, then the software has changed. Removal of software is indicated by absence of executables. It is still difficult to detect the version of the software, though this can often be done by invoking the package with a particular argument.

More recently, vendors of workstations have been providing software release and installation control. A good example is the Apollo RAI system. This consists of a set of tools for reading software from media and installing software onto workstations. Software is initially installed in a specified location (called the Authorized Area). From there, it can be installed on any other workstations on the network. The authorized area contains information on what software is available to be installed, and what the most recent update is. It would be easy to add some code that kept knowledge about where and when software was installed. This knowledge could then be acquired by FUSION.

Processes

Finding out the processes that are running on a single workstation is a well established technology. The UNIX 'ps' command is a simple example. This can provide information on the name of the process, the time it has been running, and the current state. Most networks have methods to find the processes on remote machines. Again from UNIX, the 'rsh' command allows execution on a remote machine. Since the output is directed to the current machine, it can be used to execute a 'ps' on the remote machine.

Loading measurement is also well established. The usual method is to have a continually running process (daemon) that broadcasts the current load of the machine at specified time intervals. An example is the TCP/IP 'rwhod' daemon that updates files used by the 'ruptime' command. This command gives information about the state of available machines on the network, including the approximate loading factor of the machine. It is not as good predictive expert system, however it will be considerably simpler.

Hardware

This is one the most difficult aspects to determine. It includes the processors (CPU, floating point, ...) and peripherals (disks, screen, ...) of a machine. The simplest solution is to manually enter the information in some predetermined place, which is an unreliable method at best.

Automatic detection is currently difficult. Most machines have a method to determine the current architecture, but the method tends to vary widely. For example, the Sun machines have a command called 'arch' that tells you about the current architecture. The same information is accessible on an Apollo, but requires a very different command : '/com/netstat -config'.

Another slightly more reliable method is the UNIX 'TERM' environment variable. This is usually set to the type of terminal (or machine) that the user is on. It requires some work by a system administrator, though it is a de-facto standard. The value of the variable serves as a key to the 'termcap' database. This contains information on the characteristics of the current terminal, including cursor manipulation, key bindings, and screen type.

Conclusions

As the scale and diversity of network environments increase, users will become less certain about how to accomplish tasks. Some form of system to aid in user task planning and completion is needed. FUSION will do this by providing both intelligent advice and cognitive support.

To do this requires knowledge of both the system and the user. Some sources of knowledge are outside the current bounds of knowledge acquisition techniques, so new methods of acquisition using current technologies have been suggested.

FUSION is an integration of five areas : knowledge based systems, advice giving, knowledge acquisition, object oriented systems, and user modeling. As such it is also a heterogeneous mixture of technologies. The object oriented basis of the system provides a structured protocol for communication between the different types of knowledge. The knowledge facilitates both the giving of advice and the modeling of the user. The acquisition techniques (both traditional and new) are used to gain the knowledge.

The integration of advice systems, knowledge based systems, and new acquisition techniques provides a powerful tool that can help users, both novice and experienced, in a large heterogeneous network environment.

References

Boose, J. H. (1988). *A Survey of Knowledge Acquisition Techniques and Tools.* Proceedings of the 3rd Knowledge Acquisition for Knowledge–Based Systems Workshop, Banff, Canada, November.

Carroll, J. M. and McKendree, J. (1987). *Interface design issues for advice-giving expert systems.* Communications of the ACM, Vol. 30, No. 3, January, pages 14-31.

Durfee, E. H. (1988). *Coordination of Distributed Problem Solvers.* Kluwer Academic Publishers, Norwell, Massachusetts.

Gaines, B. R. (1988a). *A Conceptual Framework for Person–Computer Interaction in Complex Systems.* IEEE Transactions on Systems, Man, and Cybernetics, Volume 18, Number 4. July, pages 532-541.

Gaines, B.R. (1988b). *Organizational Integration: Modeling Technology, Users and Management Through an Intelligent Network Support System.* Proceedings of the Network Computing Forum, October 4-7, St. Louis, Missouri. Section 3.

Greenberg S. (1988). *Tool User, Reuse, and Organization in Command Driven Interfaces.* PhD. Thesis, Department of Computer Science, University of Calgary.

Houghton Jr., R. C. (1987) *Characteristics and Functions of Software Engineering Environments: An Overview.* ACM Sigsoft Software Engineering Notes, Volume 13, Number 1, January, pages 64-84.

Kass, R. and Finin, T. (1988a). *The Need for User Models in Generating Expert System Explanations.* International Journal of Expert Systems, Volume 1, Number 4, pages 345-375.

Kass, R. and Finn, T. (1988b). *A general user modelling facility.* In Proceedings of the ACM SIGCHI '83 Human Factors in Computing Systems, pages 54-57, Boston, December 12-15.

Levi, K., Perschbacher, D., Shalin, V. (1988). *Learning Plans and Information Requirements for Pilot Aiding.* Proceedings of the 3rd Knowledge Acquisition for Knowledge–Based Systems Workshop, Banff, Canada, November.

Linster, M. (1988). *A Critical Look at KRITON.* Proceedings of the 3rd Knowledge Acquisition for Knowledge–Based Systems Workshop, Banff, Canada, November.

NCF (1988a). *Report of the User Requirements Interest Group.* Proceedings of the Network Computing Forum, October 4-7, St. Louis, Missouri. Section 1, pages 2-7.

NCF (1988b). *Report of the User Interface Working Group.* Proceedings of the Network Computing Forum, October 4-7, St. Louis, Missouri. Section 1, page 16.

Norman, D. A. (1984). *Stages and levels in human-machine interaction.* International Journal of Man-Machine Studies, Vol. 21, pages 365-375.

Pasqule, J. (1988). Using Expert Systems to Manage Distributed Computer Systems. IEEE Network special issue on expert systems in network operations and management, September, pages 22-28.

Rissland, Edwina L. (1984). *Ingredients of intelligent user interfaces.* International Journal of Man–Machine Studies, Vol. 21, No. 4, October, pages 377-388.

Smith, D. C., Irby, C., Kimball, R., Verplank, B., Harslem, E. (1983). *Designing the Star User Interface.* In Integrated Interactive Computing Systems, P. Degano and E. Sanderwall Eds., North Holland.

Taylor, M. M. (1988). *Layered protocols for computer-human dialogue. I: Principles.* International Journal of Man-Machine Studies, Vol. 28, pages 175-218.

Woodward, B. (1988). *Knowledge Acquisition at the Front End: Defining the Domain.* Proceedings of the 3rd Knowledge Acquisition for Knowledge–Based Systems Workshop, Banff, Canada, November.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

MENTAL MODELS IN THE KNOWLEDGE ACQUISITION PROCESS

Mildred L. G. Shaw and J. Brian Woodward
Knowledge Science Institute
Department of Computer Science
University of Calgary
Calgary, Alberta,
Canada T2N 1N4

email:—mildred@cpsc.ucalgary.ca

ABSTRACT

The main difficulties in knowledge acquisition from domain experts stem from the variety of types of knowledge, the various representations of knowledge, and the intrinsic problems in making these explicit and accessible. In considering these problems it is appropriate to draw on studies in cognitive science and associated disciplines to examine the style of the expert as well as the task demands and goals. The knowledge may be *operative* in that the expert can perform effectively; *tacit* in that the expert cannot express the basis of the performance; *explicable* in a way meaningful to others; essentially *dynamic* in involving continual change; *meta* in that the expert understands the extent of his or her knowledge; and so on. The concepts of *mental model, conceptual model* and *model of conceptual model* have been used to account for the human activities underlying knowledge acquisition and many systems have been developed as a result of the partial application of particular concepts of modeling. This paper reviews several methods of knowledge acquisition, some manual and some of which have been wholly or partially automated, in the context of mental models.

INTRODUCTION

The field of knowledge acquisition lacks a methodological framework which logically supports the process of knowledge acquisition, identifies the characteristics of the products of knowledge acquisition, and offers a set of ordered assumptions about the nature of knowledge and its characteristics. It is imperative that the correct or "operative" knowledge (Johnson, 1986) be identified and made explicit through knowledge acquisition; failing this, the resultant knowledge model may be a poor replica of the original.

Making the knowledge explicit in all its forms challenges the knowledge engineer. Beginning with the domain problem, the initial objectives of knowledge acquisition are to identify both the characteristics of the task situation and the method used by the problem solver to address the task. Without the determination of these two general areas of knowledge, the task of the knowledge engineer becomes an undirected search for whatever knowledge appears to fit the operational formalism chosen for the final computer knowledge base.

If the task demands can be identified and the problem-solving method of the expert ascertained, the types of knowledge required become identifiable. Procedures can then be developed to elicit these types of knowledge, either new procedures or, more likely, those borrowed and modified from other disciplines. The next four sections identify the problems of current approaches and the underlying philosophical, conceptual and methodological issues.

THE "PATCHWORK" APPROACH TO KNOWLEDGE ACQUISITION

Methodologically, the field of knowledge acquisition is characterized by a variety of methods and techniques, mostly imported from other disciplines and adapted for use in knowledge

acquisition. Often a technique has been imported to address a specific issue or problem and then its use is generalized for other purposes. Psychology has offered entity-attribute grids (Shaw & Gaines, 1987; Boose & Bradshaw, 1987) protocol analysis (Ericsson & Simon, 1984) and some forms of interviewing (La France, 1987). Linguistics has offered implicit knowledge structures for text analysis (Woodward, 1988) and forms of discourse analysis, ontological and conceptual analysis (Belkin, Brooks & Daniels, 1986; Hirst & Regoczei, 1989). Epistomology has offered knowledge hierarchies (Hayward, Wielinga & Breuker 1986). The machine learning field has provided induction techniques (Forsyth & Rada, 1986). Artificial intelligence has provided a number of knowledge representation formalisms, inference procedures, and the development of generic structures (McDermott, 1988; Chandrasekaran, 1988), and more procedures are being brought from the area of systems analysis (McGraw & Harbison-Briggs, 1989).

As each method was incorporated into knowledge acquisition, it brought with it assumptions about the nature of knowledge and how knowledge was represented. The assumptions are often implicit, but usually begin to emerge in the knowledge representation language (e.g. entity-attribute grids assume that distinctions are critical primitives underlying knowledge), but also assumptions often remain implicit as in some forms of interviewing. Do the knowledge representations actually represent the knowledge if all assumptions about the knowledge have been left unstated?

The assumptions made about knowledge are reflected in the procedures of the knowledge acquisition method or technique. Methods which emphasize the use of 'task-like' situations (e.g. think-aloud protocols) which provide prompts and cues to draw out or elicit responses are assumed to provide the required knowledge. Methods which emphasize the labeling, selecting, coding or reordering of linguistic material (e.g. discourse analysis, KADS) are assumed to provide an *a priori* structure or schema. The assumptions about knowledge contained in the structure activity also require elucidation. What parts of the knowledge are captured by these methods and what parts are not?

Borrowing methods from other disciplines has helped address the growing knowledge acquisition problem for knowledge based systems and has helped to provide a variety of useable techniques and methods. However, this approach has also resulted in the knowledge acquisition area resembling a patch-work quilt whereby methods and techniques are strung together in a pragmatic rather than systematic fashion. One may ask whether the elicitation, analysis and implementation processes make inconsistent, even conflicting, assumptions about knowledge.

Due to the lack of a systematic framework for knowledge acquisition it is extremely difficult to organize and arrange the proper and appropriate application of the techniques and methods for knowledge acquisition. When and why to apply a particular technique is often left up to the predelictions of the knowledger engineer or the availability of a program for that method.

Some researchers have attempted to solve the problem by extending those methods imported from other areas or by developing a very precise, well-formalized and very powerful method. An example of the first solution is the KADS development. Particular assumptions reflecting the epistomological underpinnings of knowledge representation were used to develop a complex system of knowledge analysis. Examples of the second solution are the knowledge acquisition methods developed by Chandrasekaran (1988) and McDermott (1988). These researchers developed specific methods for identifying repeatable activity during the solution of certain types of problems. These repeatable procedures were then formalized to identify the knowledge required which acts as a basis for knowledge acquisition.

Twine (1989) offers a suggestion for another solution to the "patchwork" problem. He proposes that we begin by making a very clear distinction between "method" and "methodology". A methodology requires the development of a set of guidelines, assumptions and eventually

principles which bring order to a field. In the case of knowledge acquisition, a clearly defined methodology would provide a set of criteria for identifying, ordering and comparing the variety of methods for knowledge acquisition. It would also provide a framework for identifying the "gaps" in knowlege acquisition techniques and methods. Once a gap was found, the methodology would aid in the principled development of new tools which meet methodological requirements. Finally, the methodology provides a framework for logically supporting the knowledge acquisition process, for identifying the characteristics of the products of knowledge acquisition and for making decisions about the nature of knowledge and its characteristics. In this paper we develop a general methodological framework for knowledge acquisition methods that clarifies their underlying assumptions and classifies their structures.

PHILOSOPHICAL ISSUES

The primary philosophical issue in knowledge acquisition may be seen as the contrast between the rational/objectivist approach to knowledge versus the constructivist approach (Woodward 1989). A rational/objectivist approach defines a problem as the observable gap between a given goal or predefined standards of performance and the present state of affairs (Nystedt, 1983). The nature of knowledge is such that knowledge is a body of information acquired by people, something that can be transferred, borrowed, stored, displayed, packaged and bought or sold like any other commodity. Within this framework, the 'task' is an objectively defined set of situational demands which must be met or addressed in order to close the gap. Assumptions are that the gap can be objectified, that the problem-solving process addressing the task can be identified and agreed upon and that the problem can be decomposed into sub-problems. Diagnostic tasks in one domain should be the same as diagnostic tasks in other domains. This philosophical position ignores the presence of the problem-solver; or at the very least, reduces the problem-solver to a container for replicable problem-solving procedures.

In contrast, the constructivist approach defines a problem as an artifact of cognitive processes (Nystedt, 1983) and problem-solving is defined as problem-formulation combined with the steps taken to arrive at cognitive closure (e.g. satisfaction, reduction of dissonance). Problems may be sub-divided but more likely they are reformulated. This approach assumes that the goal may be shared with others but there may be numerous individual methods of reaching the goal. Different experts in the same domain may use different problem-solving methods to solve the same problem. This philosophical position reflects a view which recognizes the problem-solver as central to problem-solving. The problem-solver is the producer of the knowledge. For knowledge acquisition, the focus is on identifying the features of the task which are salient to the problem-solver and making explicit the cognitive procedures which are used. Knowledge about the task is only used as it is seen through the eyes of the problem-solver (Compton & Jansen, 1989).

Both approaches assume that tasks and methods may be classified based on various qualities and characteristics. This is useful for knowledge acquisition. If knowledge engineers can recognize the *task characteristics* and the *problem-solving method* of the expert and match the two, then knowledge base development takes on a more methodical, less idiosyncratic flavor (Welbank, 1987). The salient types of knowledge can be identified and algorithms for the processing of the knowledge constructed or selected. However, there does not yet exist a taxonomy of either task characteristics or problem-solving methods which can assist knowledge engineers throughout the knowledge acquisition process (Woodward, 1989). Also, there is no useful scheme for mapping specific task demands with problem-solving processes.

CONCEPTUAL ISSUES

The conceptual difficulties in knowledge acquisition stem from a lack of consensus on what framework can be used for representing the task demands and problem-solving method (Clancey,

1989). The concept of *mental models* has been developed by a number of researchers in cognitive science. Johnson-Laird (1983) operationalizes this concept as logical propositions represented by quasi-logical propositions. This approach recognizes mental models at the 'micro' level whereas Graesser and Clarke (1985) identify "general knowledge structures" as unique and self-contained mental models. This same level of operationalization is demonstrated by other examples of general knowledge formats or packages such as frames (Minsky, 1975), scripts and/or plans (Schank & Abelson, 1977), productions (Anderson, 1983), memory organization packages (Schank, 1982), and thematic abstract units (Winston, 1984).

At the 'macro' level, the mental models concept can be operationalized as styles of problem-solving. Styles refer to the general approach taken by the problem-solver in addressing a variety of tasks. An example of this level of definition is the formalizing of modes of inquiry. Churchman's (1971) book on inquiry identifies five major schools of western philosophy which provide the basis for different types of problem-solving systems. Each system is characterized by its definition of knowledge, how the knowledge is combined, what constitutes a solution and the differing strengths and weaknesses of the inquiry process. Lockian, Leibnitzian, Kantian, Hegelian and Singerian inquiring systems are outlined. This scheme has been translated into a practical tool which attempts to identify an individual problem-solver's mode or style of inquiry (Harrison & Bramson, 1982).

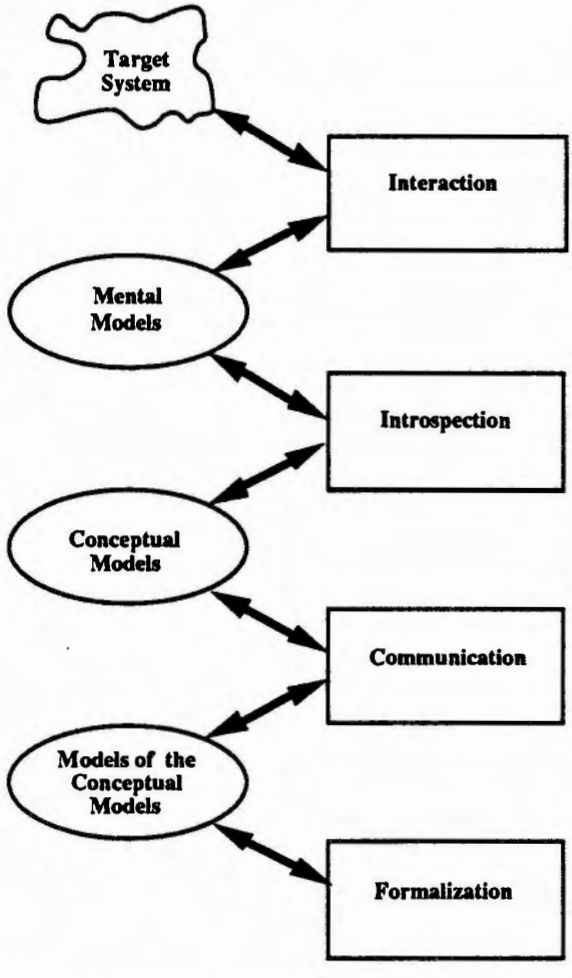


Figure 1. The modeling processes leading to the intermediate knowledge bases.

The conceptualization of mental models at the levels outlined offers useful representation schemes and operational definitions, but these lack an overall framework which would help to organize and arrange the various schemes or models of knowledge. Norman (1983) points out that the first requirement is to define the system that the expert is interacting with, "the system that the person is learning or using, by definition, the *target system.*" Figure 1 shows the relationships between the target system, the mental models of the expert, the conceptual models of the expert and the processes of interaction with the target system, introspection about the interaction, communication of the introspection, and formalism of the communication. No intrinsic model exists in the target system, but rather an ordered or repetitive sequence of activities or events. A variety of potential models are possible for this target system which represents the 'task'.

Using Norman's terminology, the expert interacts with the target system and may produce many *mental models* of the target system. The perceptual system of the expert is capable of isolating and defining 'primitives' by representing them in symbolic form. There is some debate in cognitive science about the nature of these representations and about the level at which the representations are confirmed. Empirical research suggests that it is possible for various models to exist at different levels in perceptual and cognitive processes (Pylyshyn, 1984; Fodor, 1980; Paivio, 1977).

The next modeling process results in a model of the model of the target system. Norman refers to this as the *conceptual model* of the target system because it is the model that is reflected in communicating the mental model to other entities. The conceptual model finds representation in the communication primitives of language and behavior. The mental model affects the form and representation of the conceptual model and is, in turn, constrained by the language and behavioral representational system. It is this conceptual model that is available to others and the one which requires further translation and representation by other modeling activities. As the mental model is unavailable to an observer, the conceptual model is taken to represent the model used to address the task demands of the target system.

The conceptual model becomes the focus of further modeling efforts. Knowledge engineers, or automated knowledge acquisition systems, take these models and enforce an order through the underlying models implicit in their elicitation procedures from the introspection of the expert about his interaction with the target system. The conceptual model provides a set of primitives which are communicated to the analysis processes and recognized by the analytic methods used (which, in turn, are directed by models of their own). The *intermediate knowlege base* is produced from the conceptual model through the elicitation and informal analysis procedures which rely on the introspection and communication of it from the expert. There is no direct link between the mental and conceptual models, between the cognitive processes, or between the elicitation and analysis procedures, but the processes are iterated back and forth, the models providing new data for the procedures which in turn select parts of the essential features from the models until the intermediate knowledge base emerges.

The objective of this modeling activity is to select a set of primitives from the conceptual model which can be used for another model. This activity requires an ordered and systematic procedure of its own. Representing the primitives of the conceptual model necessitates another medium and another representation structure. What is generated is a series of models of the conceptual model of the target system, in various forms of refinement and representation. These models are usually designed to represent different primitives which are considered important to the task. Primitives, which represent facts of the domain, are included with the inferencing procedures. Also, forms of strategic processes which guide the use of domain knowledge primitives and the inferencing processes are included. The result of this activity is a new model of the conceptual model which is the model of the conceptual model of the target system. This model represents the interrelationships between the task demands of the target system and the model for

addressing the demands of the conceptual model. It represents the salient features of the task and its context.

Each successive model requires a representational schema, another medium of representation, and an ordered procedure for translating one representation to another. The model of the conceptual model of the target system should contain the necessary primitives to recreate the operation of the mental model (Norman, 1983). There may be many models of the conceptual model of the target system that can perform the same activity of simulating the mental model. The model of the conceptual model of the target system represents the final working or operational model. By formally analyzing this, a computer knowledge base may be identified and coded in the representation language. As in earlier stages, this process may be iterated and refined by extracting knowledge structures produced until the computer knowledge base is sufficiently well implemented to represent the model of the conceptual model of the target system as shown in figure 2.

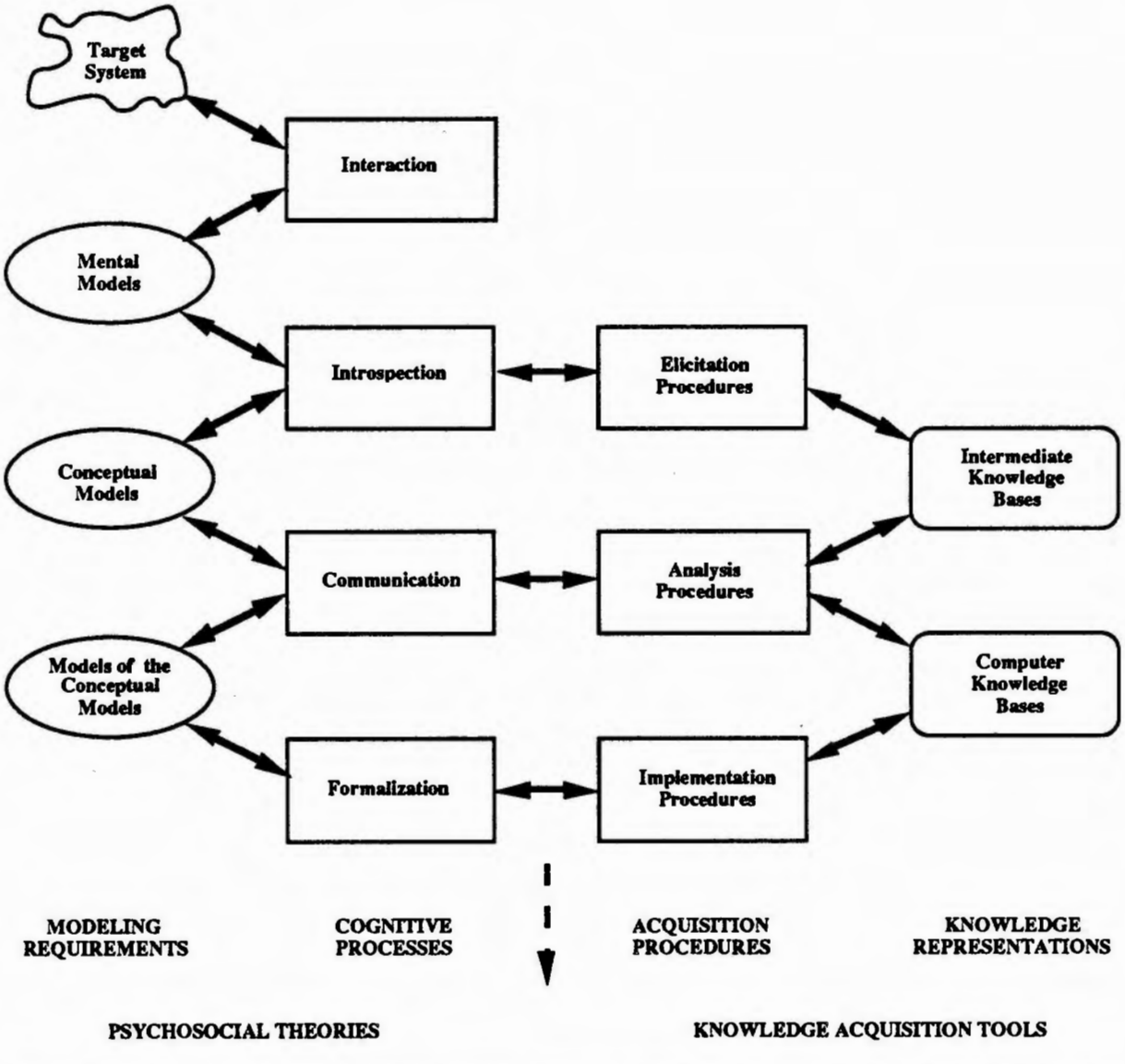


Figure 2. The processes linking the modeling requirements, the cognitive processes, the acquisition procedures and the knowledge bases

This method of development inherently links the task demands and the problem-solving method and models of the conceptual model of the target system which are developed must maintain these connections (Woodward, 1989).

The complete modeling process for the knowledge acquisition phase of development is shown in Figure 2. This shows the links from one part of the model to another and emphasizes the essentially iterative and refining nature of knowledge acquisition. The left side of the figure uses Norman's terminology applied to an expert involved a problem-solving task. It shows the social effect of communication of his problem-solving in a goal-directed situation, where the goal is to produce a computer implementation of his knowledge. The right side is shows the computer-based knowledge acquisition and representation tools.

In summary, the *target system* is the focal point for the problem-solver which has patterns of activity or knowledge inherent to the context or situation. No models exist in the target system, only ordered or repetitive sequences of events. The *mental models* represent unexpressed, uncommunicated knowledge in the expert's mind — fragmentary, partial models generated to account for his or her purposeful interaction with the target system. At this stage primitives in the situation are isolated and defined. The *conceptual models* represent the models of the models of the target system. They represent knowledge expressed in a linguistic form which can be communicated to others. they are not formal and not necessarily consistent and usually requires further refinement. They are the focus of further modeling. The *models of the conceptual models* of the target system are more formally expressed and available for communication, interpretation and discussion with others. They may encompass a variety of media and consist of a series of models which represent the interrelationships of task characteristics as modeled by the problem-solver.

METHODOLOGICAL ISSUES

Methodologically, the field of knowledge acquisition is characterized by a variety of systems, each useful for a selected type of knowledge. Confusion exists between elicitation methods and analysis methods. Norman's modeling framework also offers a way out of this methodological dilemma in knowledge acquisition. Any procedures which aid in the transition, or translation, from the mental model to the conceptual model, may be called *knowledge elicitation* methods. These procedures (interviewing, laddering, brain-storming, repertory grids, MOLE, SALT, etc.) are designed to draw out and represent forms of knowledge. In responding to the elicitation stimuli, the problem-solver arranges his or her knowledge in linguistic form (verbal and behavioral) according to the structure of the prompts. The elicitation methods are particular to specific types of knowledge and must be choosen appropriately. The conceptual model is, however, a rough model of the mental model of the target system and must be further refined into an operational form.

The translation step from the conceptual model to the model of the conceptual model of the target system can be seen as *knowledge analysis* because the end result is a knowledge model or qualitative model (Clancey, 1989) based on the linguistic primitives in the conceptual model. These methods, such as the analysis portion of KSS0 (Gaines & Shaw, 1987), induction programs (Quinlan, 1987; Dietterich & Michalski, 1981; Gaines, 1989), Cognosys (Woodward, 1988), protocol analysis (Ericsson & Simon, 1984), perform analyses on the elicited knowledge and result in another model of knowledge. These models may undergo numerous iterations. The result is a methodological distinction between elicitation and analysis procedures. This suggests the need for operationally connecting the elicitation and analysis procedures through the intermediate knowledge base which is done, for instance, in KSS0 resulting from the earlier work of Shaw (1980).

Integration of a flexible range of tools is of paramount importance (Gaines, 1987) especially with automated knowledge acquisition methods. With current systems, the knowledge engineer is allowed few degrees of freedom. If the current domain does not conform to the tool, the domain is sometimes 'redefined' to fit the tool. Manual methods, on the other hand offer a flexibility and adaptability not found with automated tools but frequently offer help with only a part of the elicitation and analysis tasks required for knowledge acquisition.

PROPERTIES OF MENTAL MODELS

Assumptions made about the mental models are propagated to later models. A simple formulation of the components required for mental representational activity come from the description of verbal reports (Ericsson & Simon, 1984) which are considered as reflecting a subset of the information held in the short-term memory. Cognitive processes are postulated as a sequence of internal states successively transformed by a series of information processes. Also, information is assumed to be stored in several different memories which reflect different capacities and display different capacities and accessing characteristics (p. 11).

The mental model, or the cognitive representations in the human knowledge source, represents that which knowledge engineers attempt to discover and model. These models represent two different processes. The first is the forming of models. Models are formed during information gathering when "some basic level concepts and relationships [are selected] on the basis of perceptual input and prior expectations" (Gilhooly, 1987 p. 23). New situations are initially categorized in terms of an existing model but if this one is insufficient then a new model is developed (Kelly, 1955) from basic concepts or by combining parts of existing models. The second use of models is in prediction and problem solving. The models suggest a set of hypotheses about the system under study. The hypotheses change as new information is perceived and incorporated (Kelly, 1955).

For knowledge acquisition, the nature of these models requires elucidation because elicitation techniques often assume (implicitly or explicitly) particular cognitive structures and processes. For example, the generic task approach (Chandrasekaran, 1988) assumes that problem-solving can be modeled as a series of "modular" processing activities using specific types or forms of knowledge. Elicitation of these processes and the knowledge forms required act as the basis for knowledge acquisition activity.

In order for elicitation procedures to be more useful, the underlying assumptions about knowledge and knowledge processes require elucidation and definition. The structures and descriptions of knowledge types, roles and functions require objectification and definition to be useful to the knowledge engineer. This is accomplished when a well developed theory underlies the elicitation (and hopefully the analysis method as well) and is used to make assumptions about the structure of knowledge. KSSO (Gaines & Shaw, 1987) is based on Kelly's (1955) method of repertory grids within the framework of personal construct psychology. Cognosys (Woodward, 1988) uses the method of generic knowledge structures (Graesser & Clark, 1985) as a method within the framework of implicit knowledge structures. These methods make very definite assumptions about the nature of the knowledge and the inference processes at play in the mental model. Other researchers (Craik, 1943; Johnson-Laird, 1983; Gentner & Gentner, 1983; Norman, 1983) use the concept of mental models to outline and define the components of knowledge. It has also been postulated that various mental models exist at various levels in the perceptual and cognitive faculties (Pylyshyn, 1984; Fodor, 1980; Paivio, 1977).

PROPERTIES OF INTROSPECTION AND ELICITATION PROCEDURES

Elicitation techniques for knowledge acquisition support introspection of the expert and act as classification grammars for linguistic elements and linguistic levels. After the task situation has

been selected, instructions are normally given to the expert in order to generate verbal and/or behavioral events. These events are collected, then ordered or parsed into useful phrases which form the conceptual model. The specific instructions act as a constraint on the mental model material in that they structure the verbal material from the expert. The method of collection may result in fairly complete collection such as video recording or partial collection such as note-taking.

There appear to be few variations on the main types of elicitation procedures. Each type involves assumptions about the type of knowledge or information and where it resides. Ericsson and Simon (1984) note that information can be elicited retrospectively, inferentially, descriptively, or by direct questioning. In the first case, retrospective elicitation is assumed to tap information stored in the short-term memory if the time between the completion of the task and the elicitation procedure is not too long. It is assumed that the expert can describe the heeded information used for task completion.

When given instructions to report on the steps in solving a task, the expert will make inferences about what he or she 'must' have thought. Inferentially generated material in elicitation procedures constrains the expert's responses by requiring information that is plausible. The information tapped in this case may or may not have anything to do with the heeded material or those salient knowledge cues used in completing the task. Rather, the communication context between the knowledge engineer, or knowledge acquisition tool, and the expert becomes more salient.

Descriptive types of elicitation take the form of thought descriptions at the time of completing the task. These types of elicitation instructions promote the verbalization of information which is currently in short-term memory. It is assumed that the verbalization processes do not interfere with the completion of the task or influence the processes required for task completion (Ericsson & Simon, 1984). The vocalizations are assumed to be a commentary on the problem-solving method used while the expert is completing the task.

Direct probing is assumed to elicit pieces of information directly from the short-term memory with no need for intervening accessing cues. If the information is not in short-term memory, then the direct probing can still be used to elicit information in long-term memory by way of accessing cues. La France's (1986) grid is based on the assumptions that information is stored in many different memories and so can be accessed using a variety of probes.

The information resulting from the above elicitation types may reflect different qualities of verbal material. The simplest verbal material in the conceptual model results from direct verbal descriptions of the information heeded while completing the task. Very little effort is required by the expert for this activity because the information points within the awareness of the expert while completing the task are simply labelled.

More complex material comes from elicitation into the conceptual model when the expert begins to describe the thought content of the mental model in short-term memory. This information assumes an 'observer' role such that the expert is able to describe the thought processes and knowledge roles which are salient and relevant to the task. It is assumed that these verbalizations are descriptions of the mental model in action or descriptions of the mental model in past action.

The most complex level of verbalization occurs when the expert begins to explain the thought processes, thoughts, ideas, hypotheses or motives required when completing the task. The act of explanation requires the linking of cues in short-term memory to earlier information and/or to information in long-term memory. This need to link information not currently in the short-term memory results in an increase in the amount of information and may even after the salience and relevance of the information (Ericsson & Simon, 1984, p.79).

Elicitation procedures implicitly assume characteristics about the knowledge they intend to elicit; they select and order some of the information available in the mental model. The elicitation procedure uses its 'theory' to constrain the types of information from the expert. For example, KSS0 asks the expert for objects or "entities" which the expert believes pertain to a certain purpose or task. These entities may be physical objects or procedures, design components, cases, etc. Then, the expert is asked to describe the distinctions between and among the objects. This form of retrospective elicitation uses its background theory to identify the required elements of the mental model and produces a severely constrained, but highly functional, conceptual model and intermediate knowledge base in the completed entity-attribute grid. The grid can then be used as the basis for a variety of analyses.

PROPERTIES OF CONCEPTUAL MODELS

For knowledge acquisition, this model is the final outcome of elicitation activity. The elicitation procedures structure the information from the expert and it is assumed that this information reflects some of the available knowledge characterizing the expert's mental representation of the particular task as it is interpreted and represented by the elicitation procedures. The conceptual model acts as the first available representation or model of the expert's thinking processes and knowledge concerning the task situation. It contains representations of communicable information about the mental model.

In most cases, the conceptual model is represented in the form of linguistic structures. Because of this representation formalism, a number of assumptions are made about the information available due to the use of linguistic structures. A basic assumption is that linguistic structures are an adequate, reliable and valid means of studying human thought processes. This assumption has been the basis for much of the modern research on linguistics (e.g. Chomsky, 1975) but has not been explicitly stated in knowledge acquisition work. It is possible to extend this main assumption to make it more applicable to knowledge acquisition by outlining a number of assumptions about knowledge and cognitive processes which apply in knowledge acquisition work.

In linguistics it is postulated that the linguistic material produced is a surface-structure representation of a base-grammar, that operations are performed on the base grammar to produce a communicable description of the underlying representation. The view of transformational grammarians (e.g. Chomsky, 1975; Gazdar, 1981; Bresnan, 1982) is that deep structure relations are transformed through a semantic analysis to produce surface, or linguistic, representations. Elicitation methods appear to affect the integration processes of the transformations, and depend upon this transformational process in that such procedures constrain the types of surface-structure components and relations possible.

It is the nature of linguistic material which allows the knowledge engineer the opportunity of making inferences about the material. Graesser and Clark (1985 p. 14) suggest that much of the meaning of the conceptual model is not exclusively in the text. Inferences can be drawn based on contextual cues, on the linguistic material itself, on the goals of the comprehender, on the degree of shared knowledge between the generater and reader of the conceptual model and, finally, about the knowledge structures in long-term memory. The richness of the conceptual model depends somewhat on the elicitation process, and the usefulness of the linguistic representation of the conceptual model lies in the knowledge engineer's ability to generate meaning based on properties of the linguistic structure.

It must also be assumed that the linguistic material somehow reflects abstract structures in the mind which make knowledge possible. The production of linguistic structures are assumed to reflect these mind-based abstract structures and processes. That knowledge engineers and

knowledge acquisition methods are able to aid in the linguistic description of internally represented structures and processes has not been proven experimentally but is assumed to be true. We believe that what we elicit is an approximate representation of knowledge; assumptions are made about the ability of linguistic material to capture the expert's intrinsic competence and internalized knowledge.

Ericsson and Simon (1984, p. 221) make specific assumptions about the cognitive mechanisms underlying verbal descriptions. They state that:

1. "verbalizable cognitions can be described as states that correspond to the contents of short-term memory (i.e. to the information that is in the focus of attention);
2. " the information vocalized is a verbal encoding of the information in short-term memory;
3. " The verbalization processes are initiated as thought is heeded;
4. " the verbalization is a direct encoding of the heeded thought and reflects its structure;
5. " units of articulation will correspond to integrated cognitive structures; and
6. " pauses and hesitations will be good predictors of shifts in the processing of cognitive structures."

PROPERTIES OF INTERMEDIATE KNOWLEDGE BASES

Most knowledge engineers, and knowledge acquisition systems, produces multiple and varied intermediate knowledge bases. Welbank (1987) interviewed a number of knowledge engineers and discovered that there is much variety in how knowledge engineers deal with linguistic material from the conceptual model. Idiosyncratic, ad hoc, methods were reflected in the use of a variety of diagramatic methods and representation procedures. There is more activity in the knowledge acquisition community which has focused on the production of procedures for generating intermediate representations of the conceptual model material (e.g. Johnson & Johnson, 1987).

Intermediate knowledge bases are best characterized in terms of their functions. One main function is as a representation for display to the expert so that validation and confirmation are possible. A diagram is useful to help confirm understanding of the expert's mental model. A second function is to act as a focal point for further elicitation; the initial elicitation acts as the foundation for acquiring more information. An example is the development of a domain dictionary composed of term definitions (McGraw & Harbison-Briggs, 1989) which constitutes a domain vocabulary. Another example is the development of ontological categories based on the syntactic position of words. A noun suggests a static object; a verb suggests action and change; adverbs and adjectives suggests styles and distinctions. Attributes or distinctions can be elicited and combined in groups.

Intermediate knowledge bases are characterized by structures and forms based on qualities already inherent in the natural language structure of the conceptual model rather than on a formalized analysis procedure. Because of their functional properties, intermediate knowledge bases are informally-produced, partially-ordered, linguistically-based, and inherently unstable structures.

PROPERTIES OF COMMUNICATION AND ANALYSIS PROCEDURES

Informal analysis procedures support the expert's first attempts at communication and draw out meaning from the linguistic material of the conceptual model and the partial orderings and structures of the intermediate knowledge bases. Informal analysis procedures are used to identify underlying, implicit or assumed knowledge structures in the elicited material. These procedures are usually theory-laden in that they reflect assumptions made about the structure of knowledge and how the structures can be operationalized. These analysis procedures are possible only because of an underlying theoretical framework which justifies the labelling and ordering of the

material. They are possible also because of their formality and consistency. They reflect a step-by-step, algorithmic nature and they are capable of being applied consistently to elicited material.

Analysis procedures usually begin with some form of encoding of the linguistic material from the conceptual model or from the intermediate knowledge bases Encoding of these materials impose an *a priori* framework over the conceptual model and intermediate knowledge base material. This framework identifies the 'required' linguistic primitives and labels or categorizes them. For example, in the KADS method (Breuker & Wielinga, 1987), verbs or groups of verbs in the text are viewed as "knowledge sources" in the sense that verbs indicate action and that action is usually done to something to produce something else. Knowledge sources are considered as producers of structure or meaning. Nouns or groups of nouns are, in turn, considered as objects which function as either inputs to knowledge sources or as the products or outputs of knowledge sources.

Once material has been coded, more formal analysis techniques group the encoded material into various structural configurations which aid communication of the material to other people and processes. Groups of encoded material are distinguished which can then be ordered, compared, matched, or constructed to produce more meaningful knowledge structures. In Cognosys (Woodward, 1988) "knowledge units", noun-verb-noun phrases, are labelled as events, states, goals or styles. These are then connected to other knowledge units on the basis of linking rules (Graesser & Clark, 1985). Only knowledge units of certain types can be linked to others. Once the linking is complete, the categories are used to develop a variety of sub-structures which appear evident in the task. For example, goal-oriented sub-structures identify the ultimate goal and sub-goals. Implication sub-structures identify implicit connections within the task.

PROPERTIES OF MODELS OF THE CONCEPTUAL MODELS

The results of formal analysis procedures produce a well-defined, conceptually ordered, theoretically consistent model of the task and context under consideration. This model can be viewed as a completed theory of knowledge, based on consistent use of concepts and a methodical application of formal rules which is able to trace the origins of the knowledge and document its transformations. The model of the conceptual model stands as the formalized representation of the more expressive conceptual model.

The model viability is ensured by containing all the necessary elements which allow the implementation process. Addis (1989) has identified three necessary components of knowledge which may be applied to the models of the conceptual models: first, each model should contain the required declarative knowledge reflecting tacit and factual knowledge inherent in the problem-solving method and in the task demands; second, the inference knowledge which operates on the declarative forms is necessary to identify the transformation or processing steps of the knowledge; third heuristic knowledge is required which sometimes act as special constraints, sometimes as inferences and sometimes as strategic procedures.

This model of the conceptual model is also characterized by formal interrelationships between task demands, external to but perceived by the problem solver, and the problem-solving method used by the problem-solver (Woodward, 1988). Task cues and their meanings are represented in the model and these cues are directly linked to inference, heuristic and declarative structures.

PROPERTIES OF FORMALISM AND IMPLEMENTATION PROCEDURES

Implementation processes draw upon the formalized structure of the model of the conceptual model. Based on well established principles of artificial intelligence and underlying assumptions about knowledge, these processes select, order, arrange and represent elements of the model in a form conducive to operation in a computer-based knowledge system. Inference, heuristic and

declarative knowledge are all coded into formats which reflect their different natures and functions. A variety of declarative knowledge forms are used to build portions of the knowledge base. Inference knowledge is represented by the ways in which the declarative knowledge is processed and manipulated. Heuristic knowledge is represented as meta- or control knowledge.

Due to the number of possible formal methods of implementation, different aspects of the model of the conceptual model may be selected. Rule-format coding differs from frame-based coding which differs from logic-based coding. The appropriate choice depends on whether the required knowledge structures are represented in an applicable form within the model of the conceptual model. An efficient model would allow many implementation methods.

PROPERTIES OF COMPUTER KNOWLEDGE BASES

This final result reflects the multiple methods used in the process. The computer knowledge base represents a theory of knowledge that was constructed through the elicitation, analysis and implementation processes and based the assumptions underlying the representations in each modeling phase. Declarative knowledge is usually represented separately from the inferential knowledge but once complete, the final knowledge base operates in algorithmic fashion.

The first part of the paper has identified the primary characteristics of and knowledge acquisition process to be the underlying models and presuppositions they make, and the elicitation, analysis and implementation procedures used. The next section analyses some widely used knowledge acquisition techniques in these terms. The table below summarizes the techniques surveyed, how they relate to the presupposed models and the procedures used in the elicitation, analysis and implementation techniques used in each case.

Method	Presupposed Model	Elicitation Procedures	Analysis Procedures	Implementation
Interviewing (La Franco, 1987) (Johnson & Johnson, 1987)	General but weak	Interview: unstructured semi-structured structured	Extraction and organization of domain concepts	Not supported
Protocol Analysis (Ericsson & Simon, 1984) (Johnson, Zualkernan & Garber, 1987)	Possibility of adequate introspection	Recall of introspection	Extraction and organization of domain concepts	Not supported
Empirical Induction (Quinlan, 1987) (Dietterich & Michalsky, 1981) (Gaines, 1989)	Domain structure already identified; large dataset of expert behavior available	Collect large dataset of expert protocols (evaluations and decisions)	Induction of decision rules	Backward chaining rule-based system
Domain Ontologies (Breuker & Wielinga, 1987)	Applicable multi-level ontology for knowledge representation	Structured interview; protocol analysis	Fit ontological structure to domain	Logic-based interpretation of structure
Problem-Solving Ontologies (McDermott, 1988) (Chandrasekaran, 1988)	Existence of applicable generic problem-solving methods	Elicit repetitive process of problem-solving in domain	Fit generic problem-solving methods to observed behaviors	Parameterize implementation of generic problem-solving methods
Implicit Knowledge Structures (Woodward, 1989)	Implicit knowledge structures	Recall expert problem-solving behavior in domain	Identification of noun-verb phrases	Use as a basis for semantic net, frame or rule-based coding formalisms
Entity-Attribute Grids (Gaines & Shaw, 1987) (Boose & Bradshaw, 1987)	Knowledge characterized by distinctions made	Elicit entities in domain and distinctions between them	Conceptual clustering to confirm domain structure; induction to determine constraints	Generate classes, properties, methods, objects and values

Table 1. Some major knowledge acquisition techniques classified according to the presupposed model assumed, and the procedures used for elicitation, analysis and implementation.

SOME MAJOR TECHNIQUES FOR THE ACQUISITION OF KNOWLEDGE

Interviewing (McGraw & Harbison-Briggs, 1989) is used to identify tasks and major concepts, to structure and refine already-acquired information. Unstructured interviews are appropriate when the knowledge engineer wants to explore an issue or the goal is to establish rapport and to get a broad view. It facilitates the description of the domain in a way that is easy for the expert to understand. However, the data acquired is often unrelated and difficult to integrate due to its lack of structure. It may not allow gathering of specific knowledge and takes time and training to do well. Also, similar questions may be asked in future sessions and this may annoy the expert. Alternatively, structured interviews have the advantages of forcing an organization on the interview and being very goal-directed. They attempt to remove distortion from experts subjectively and allow better integration of material after the interview. They force the expert to be systematic, and the knowledge engineer is able to identify gaps in the knowledge which act as a basis for further questions. The purpose of the session is clear to the expert. However, they need more preparation by the knowledge engineer who must study the background material extensively.

During the interview, of any kind, the knowledge engineer must decide on a note taking approach, for example, video- or audio-taping, each of which has advantages and disadvantages. Both open and closed questions can be used. Open questions are broad and place few constraints on the responder. They are used for determining scope of understanding, response certainty, and the models used. These questions like *what, why, how* allow the expert to express information the knowledge engineer does not know about. He can obtain the expert's vocabulary, concepts, frames of reference, and can help with explanations and underlying theory. Closed questions such as *who, where, when, which* tend to set limits on the type, level and amount of information the expert provides since they often provide a choice of alternatives such as a bipolar or multiple choice response. These are used more for clarifying or probing questions or as feedback as they are less time consuming for specific information. Note-taking is much easier but sometimes one can get too little information as they may stop the expert from volunteering information. They requires an excellent command of vocabulary and concepts from the knowledge engineer. On balance a combination of open and closed questions is preferable.

One example of semi-structured interviewing is the Knowledge Acquisition Grid (LaFrance, 1987). This is a matrix of knowledge types and forms where the forms of knowledge are *layouts, stories, scripts, metaphors* and *rules-of-thumb*; and the question types are *grand tour, cataloging categories, ascertaining attributes, determining inter-connections, seeking advice* and *cross-checking*.

Examples of a "grand tour" are distinguishing boundaries of the domain, perspective, goals organization, e.g. Please describe the kinds of things you do; of "cataloging may be an organized taxonomy of expert terms and concepts, e.g. How would you group and subgroup these things?; of "ascertaining attributes" are distinguishing features and range of values, e.g. Describe similarities and differences of concepts; of "determining interconnections" are uncovering relations and causal model(s), e.g. Why?; of "seeking advice" include revealing strategies, e.g. Which conditions warrant which actions?; and of "cross-checking" are validation concerns, e.g. knowledge engineer plays naive, devil's advocate, poses hypothetical situations and asks How sure are you? Are there exceptions?

The teachback technique of Johnson and Johnson (1987) is a semi-structured interviewing method based on Pask's (1975) teachback techniques and Systemic Grammar Networks from Linguistics. This method attempts to formalize a functional approach to grammar. It is an intermediate representation used to teach back to the expert the knowledge acquired in the interview. It results in a mediating representation which is a machine language independent

formalism between verbal data and software development environments. It is used as a tool to help proceed on early elicitation and conceptualization for synthesizing knowledge from talk with experts. The expert is encouraged to use his or her own words to express facts, procedures and abstractions which are recorded. The transcript is segmented, paraphrased, indexed and annotated by the knowledge engineer and the results captured in a systemic grammar network and checked and amended with the expert. It is mainly used as a means to prepare explicit descriptions of knowledge for the use of the development team to construct an associative network, graphs composed of nodes joined by links. It is said to be a generative form which captures all relevant information but excludes absurdities.

Interviewing protocols whether structured or unstructured, specific or general, probing or expansive, usually require a retrospective approach by the expert. This method makes implicit assumptions about the processing of knowledge but little about the nature of knowledge. Interviewing assumes that the knowledge expressed by the expert is based on the question prompts and is the actual knowledge used by the expert during task completion. Further, this knowledge needs to come into short-term memory before the expert can answer and that the expert needed to use some set of accessing cues (La France, 1987) to bring the knowledge from long-term memory. This process is vulnerable to influence by the structures of the communication situation between expert and knowledge engineer.

This method assumes a neutral stance with respect to types of knowledge elicited, although the question content itself makes assumptions about what knowledge is important and required. The encoding procedures developed for the results of interviewing (e.g. discourse analysis) are left to provide some implication of the nature of the knowledge acquired.

The result of interviewing is a very rich, but extremely unstructured and perhaps somewhat invalid (Bainbridge, 1986) conceptual model. On the other hand, the conceptual model allows a great number of intermediate knowledge bases which may be developed, described, demonstrated, redefined, etc. These intermediate knowledge bases act as useful indicators of validity and as stimuli for further elicitation.

There exist few formal procedures for the analysis of unstructured material. Even though these other procedures are being adapted from other disciplines, they do not yet meet the requirements of producing a formal model of the conceptual model. The results of grounded theory and of the systemic grammar method actually result in a better conceptual model. Jumping from a conceptual model to coding processes without the aid of a formal model of the conceptual model may account for much of the difficulty and inaccuracy of the interviewing method of knowledge acquisition.

Protocol Analysis includes both observations of expert behavior and expert verbalizations. Two examples which are widely used, are Verbal Reports as Data (Ericsson & Simon, 1984) and the Specification of Expertise (Johnson, Zaulkernan & Garber, 1987). Ericsson's method uses concurrent and retrospective verbalization for elicitation. "Concurrent report" reveals the sequence of heeded (attended to) information and "retrospective report" reveals the sequence of specific cognitive processes. A think-aloud transcript is then encoded into segments (assertions propositions, reading, doing arithmetic, visual searching, long-term memory recall, generalizing combinations) and relations of code are derived from verbs, propositions and adjectives. "Arguments" are often nouns and nouns phrases and the task analysis defines *a priori* a space of possible encodings representing information relevant to the task.

The method is based on the facts that:
- heeded (attended to) information is identified by analysis of corresponding verbalization through inference and task analysis

- the original encoding should reflect as closely as possible the verbalized information, preferably by automated or semi-automated means
- encoding uses a problem space approach (behavior is a search through a problem space involving the step-by-step accumulation of knowledge)
- the model distinguishes information directly stored in memory (facts, experiences, perceived events and behavior in past situations) from information generated or produced by inference (reactions and behavior in hypothetical situations)
- the criteria for inferring underlying cognitive processes from thinking aloud protocols ar relevance to task analysis and consistency with preceding verbalizations
- information in focal attention is vocalized directly or is encoded, recovery implies decoding
- segments are analyzed using syntax, pauses and intonation
- each segment is independent (the context is ignored for the reason cited above)
- encoding involves finding a category expressing the same information as the segments

The advantages of this method are that it can provide a complete record using both concurrent and retrospective reports then after definition of problem space and segmentation of the protocol the analysis can be automated or the task made much easier for the coder as it focuses the coder's attention on the segments. It requires consistency and explicitness of underlying assumptions, vocabulary and inference rules and the automated encoding lessens the risk of incorrect inference. A comparison of talking aloud protocols shows a very high consistency for practiced skills and successive protocols show stable task strategies and cognitive representations. It is an appropriate method for reporting specific, recent, episodic memories. On the other hand, retrospective reports may leave out important detail and include inferences and generated information (plausible explanations). It is often difficult to tell if the expert pays attention to the knowledge engineers's instructions. The expert cannot report cues allowing recognition of stimuli since these are in long term memory and his short term memory retains goals and intentions longer than heeded information. There are also incompleteness problems. Thought in non-oral form is much faster, for example, the knowledge engineer cannot instruct a juggler to slow down!

The Specification of Expertise (Johnson, Zaulkernan & Garber, 1987) is one way of using the material gained from protocol analysis. It is a psychology based manual technique in which the expert thinks aloud while working. It encompasses a syntactic and semantic analysis of the protocol using operations, episodes, and data cues. Expertise is characterized by semantic categories, actions abilities, goals, conditions, strategies and solutions. It is based on the facts that thinking is not directly observable, especially that of expert with considerable compiled knowledge and practice and that expertise is characterized by generativity and is a kind of knowledge that is used to perform a task. Part of this knowledge is represented by a set of abilities for problem solving. The representation or specification of expertise consists of: possible solutions and components of solutions; relevant information or data needed; goals for solutions; permissible ways to move between intermediate solutions and the abilities needed to make a transition.

This psychologically-based method of knowledge acquisition makes assumptions about knowledge from an information processing point of view. The method assumes that an individual is able to describe the contents of his or her own short-term memory as if the individual had an internal observer watching information come and go from working memory. Ericsson and Simon (1984) demonstrate support for this method and its underlying assumptions about cognitive information processing but little is stated about the nature of knowledge apart from stating that perceptual information about environmental cues and internal cognitive processes are within the awareness of the 'observer'.

Elicitation takes the form of a variety of 'thinking-aloud' activities while completing the task of interest. As the expert thinks aloud, the entire linguistic material is recorded and usually

transcribed. The transcription can then be parsed or ordered for later coding or encoding. The conceptual model is rich and contains generated material as opposed to retrospective material (e.g. interviewing) and can be used to develop domain dictionaries and parsings based on discourse analysis procedures (e.g. Belkin, Brooks & Daniels, 1986).

Analysis procedures vary for protocol analysis because this method was developed as a generic elicitation strategy. Johnson, Zaulkernan and Garber, (1987) used a method of encoding and ordering of the protocol material. Categories such as cues, triggers, goals, activities, etc. were ordered in graphical form to give a well-defined, formally ordered model of the conceptual model. The KADS method also uses protocol analysis but uses a much different theoretical approach to analyzing the conceptual model.

A variety of coding methods are available but the choice depends on the analysis procedures used and the nature of the model of the conceptual model.

Domain Ontologies include the KADS methodology or Knowledge Acquisition Design System (Breuker & Wielinga, 1987) which deals with the whole process from domain organization to developing a complete system. It breaks the domain organization into stages where the problem is decomposed into component parts and the analysis of knowledge is separated from its machine implementation. Data is then collected and tested to modify the framework. The three refinement cycles are applied: orientation which includes the acquisition of vocabulary, the assessment of characteristics, and the selection of the model and a test for problem feasibility; problem identification in which structures of domain concepts are uncovered, the functional analysis of the prospective expert system and task analysis; and problem analysis where the analysis of the user and operational environment results in the basic architecture of the system. At this point knowledge primitives are identified such as objects, knowledge sources, structures, strategies and models. Interpretation models are produced using a top-down model-driven approach which may include notions of causality, time, space, change and task-dependent concepts.

Building interpretation models is done by knowledge identification and knowledge conceptualization where knowledge primitives are ordered using relationships such as is-a, part-of, causes, depends-on, consists-of, defines, and several types of analysis are performed. The system is then coded using guidelines laid down.

Underlying the KADS method of knowledge acquisition is a philosophically-based theory of knowledge. Epistemologically, knowledge is considered a multi-level phenomenon. Each level represents a different level of complexity and functionality (Hayward, Wielinga & Breuker 1986). They integrate the four degrees of depth at which questions about knowledge can be asked, that is, questions at the *individual, conceptual, formalism* and *mechanism* levels suggested by Sloman (1980), with the five types of knowledge *implementational, logical, epistomological, conceptual* and *linguistic* suggested by Brachman (1979). They identify the five levels as being "knowledge identification", "knowledge conceptualisation", "epistomological analysis", "logical analysis" and "implementational analysis". Knowledge cannot be considered uni-dimensional. This assumption indicates knowledge is not complete without taking into account all levels and their relationships.

Elicitation methods in KADS are designed to develop linguistically rich protocols of expert behaviour. Broader than just protocol analysis, the KADS method generates a variety of verbal protocols from specialized interviews, thinking-aloud instructions, introspection and user-dialogues. The resulting conceptual models are rich linguistic models which are parsed partially on the basis of syntactical position of the words to form domain vocabularies, diagrams, etc. and are sometimes used to identify the format of the final model of the conceptual model.

Analysis procedures (Breuker & Wielinga, 1987) follow closely the multi-level theory of knowledge. Objects, knowledge sources, tasks and strategies are all hierarchically ordered levels pulled out of the conceptual model and intermediate knowledge bases. The final model of the conceptual model is a rich and highly organized formal model called an interpretation model.

Coding procedures are represented by logic-based methods and structures. It is not yet clear how much of the richness of the model of the conceptual model survives the transformation into the computer knowledge base. Other coding methods may be possible with the rich model of the conceptual model but have not, as yet, been described.

Problem-Solving Ontologies are exemplified by *role-limiting methods* (McDermott, 1988) and *generic tasks* (Chandrasekaran, 1988), and reflect a somewhat different nature than most other methods in that they are highly structured and highly specific elicitation procedures based on a great deal of effort prior to elicitation. Both are based on the assumption that there exist repetitive processes of problem-solving used by the expert to complete a task, these methods concentrate first on identifying the repetitive roles of knowledge or those processes of knowledge use. The types of knowledge required are defined by the requirements of the processes.

Role-limiting methods strongly guide knowledge collection and encoding having a broad scope of applicability and providing help in specifying what knowledge needs to be collected to perform a particular task. A role-limiting method typically consists of a simple loop over a sequence of five to ten steps. For example, MOLE is a knowledge acquisition tool for generating expert systems that do heuristic classification. The method has the following control knowledge: determine the events that explain the symptoms; differentiate among several candidates by ruling out explanatory connections or candidate explanatory events, or by providing sufficient support for a candidate or reason for preferring some explanatory connections; get this information and apply it; at this stage, if new symptoms are revealed, go back and perform the sequence again.

Chandrasekaran is aiming to raise the level of knowledge structures from rule type systems to generic tasks which are problem-solving procedures for specific tasks. Elicitation is highly structured and specific in that an algorithm is defined for a problem and an automated elicitation mechanism is developed to structure the expert's verbalizations. If the problem-solving method is, for example, heuristic classification within a diagnostic task, then certain types of knowledge such as symptoms, hypotheses, and differentiation information are requested of the expert.

The conceptual model and the model of the conceptual model are identical in this method. Implementation methods are very specific due to the nature of the model of the conceptual model, and the computer knowledge bases which result are automatically built from the elicitation, analysis and implementation procedures.

Implicit Knowledge Structures are used in linguistics, and one method resulting from this approach is Cognosys (Woodward, 1988) which is mainly used in domain organization to structure the domain in order to isolate and organize key areas of content. The expert provides a description statement of overall domain and a list of general objectives which must be met in order to solve problems or make decisions successfully with the domain. He must then produce short paragraphs which are descriptive of the tasks required to meet each objective. These may be articles provided by the expert, such as case studies, text books or operating manuals. The knowledge engineer takes these and analyses the tasks into categories. He also completes the concept description categories and clarifies and elaborates such things as temporal relationships, event and state sequences, goal specification, cause-effect relationships, inferences, hierarchical relationships. It is important at this stage to clarify the domain boundaries with the expert and discuss specific problems in the domain. He may identify other sources of domain knowledge, before he identifies knowledge acquisition methodologies and tools, report formats and inference processes. A general knowledge structure is then produced by the knowledge engineer from

which goal structures, cause-effect structures, implicational structures and taxonomic structures are identified and isolated.

This linguistic-based method of knowledge acquisition is based on the theory of implicit knowledge structures which underlie general knowledge structures (Woodward, 1988). This method assumes that knowledge structures consist of discrete units which are linked and stored together in memory. When a word is chosen (in linguistic form), it is assumed that there exist a number of semantic meanings attached to the word. The word represents the tip of a meaning iceberg. This representation is not completely static but contains variable links which are partially sensitive to context. When words are chosen for textual material to describe, explain, or label symbolic entities, they can be used to identify the underlying or implicit knowledge structures.

This conceptual model is rich and requires further ordering but it is partially constrained by the objective-task breakdown. A variety of intermediate knowledge bases are possible. Lists of structural concepts (nouns) and functional concepts (verbs) result from the parsing of the textual material. Attributes describing each of the concepts are listed and more may be elicited. Contextual information about each concept can be stored.

The analysis procedures require the application of rules by the knowledge engineer who identifies noun-verb-noun phrases, assigns them a type and then links the phrases with a restricted set of arcs based on the meanings in the paragraphs (Woodward, 1988). These procedures follow directly from the theory of implicit structures and identify generic structures which define set-relations, goal sequences, implications, cause-effect relations and temporal sub-structures.

The set of sub-structures for each task and objective represents the model of the conceptual model. This formalized model then acts as the basis for semantic net, frame, or rule-based coding formalisms to generate a computer knowledge base.

Entity-Attribute Grids are derived from the repertory grids of personal construct theory (Kelly, 1955) which is one example of a modeling system that externalizes human processes. The personal construct theory model views the individual as an anticipatory system anticipating the future as a hypothesized replication of the past; constructs about the world change as a result of feedback about the predictions or actions on it. This is a systemic theory of human cognition based on the single primitive of a construct, or dichotomous distinction. Constructs are:

> "transparent templets which [a person] creates and then attempts to fit over the realities of which the world is composed." (Kelly 1955)

He proposes that all of human activity can be seen as a process of anticipating the future by construing the replication of events:

> "Constructs are used for predictions of things to come, and the world keeps rolling on and revealing these predictions to be either correct or misleading. This fact provides a basis for the revision of constructs and, eventually, of whole construct systems." (Kelly 1955)

Hence his psychological model of man is strongly epistemological and concerned with the way in which man models his experience and uses this model to anticipate the future. The anticipation may be passive, as in prediction, or active, as in action. Kelly developed his theory in the context of clinical psychology and hence was concerned to have techniques which used it to by-pass cognitive defenses and elicit the construct systems underlying behavior. This is precisely the problem of knowledge engineering.

A repertory grid (Shaw 1980) is a two-way classification of data in which events are interlaced with abstractions in such a way as to express part of a person's system of cross-references between his personal observations or experience of the world (elements or entities), and his personal constructs or classifications of that experience (attributes). It is a way of representing

personal constructs as a set of distinctions made about entities relevant to the problem domain. In the development of expert systems the entities might be key elements in the problem domain such as oil-well sites or business transactions, and the attributes express what the particular expert sees as the crucial distinctions between the entities in the context of the problem.

This psychologically-based method of knowledge acquisition assumes that the mental model is partially characterized by distinctions made about the target system by the expert and used to help make predictions about the world. Mental categories are present in which objects act as instances of those categories. These distinctions can be represented by bi-polar relations and objects in the world can be distinguished on the basis of the bi-polar relations.

The entity-attribute grid elicitation procedures follow very directly from the assumptions. Within a given situational or contextual system, objects are elicited which have something to do with the context. Distinctions are elicited by offering triads of objects and asking why two of the objects are the same but different from the third. The distinctions are defined by bipolar descriptions and the remaining objects are ranked upon the bi-polar dimensions depending on the extent to which they are characterized by the distinctions at one or the other end of the dimension. An assumption made by the procedure is that all necessary objects and distinctions are elicited in this manner.

The result of this activity is a very clean conceptual model but one which is limited in linguistic material. The grid does not capture all that occurs or all the knowledge that is used when completing the elicitation exercise. If one were to tape record a session while a grid was elicited while the expert worked under thinking aloud instructions, a fuller conceptual model would be obtained which could include explanations for why choices were made, the decision-making verbalizations and the descriptions of the new understanding acquired by the expert. An intermediate knowledge base becomes available from the elicitation. Any grid which results is part of the conceptual model and may act as one possible intermediate knowledge base in that it reflects a partial ordering of the material in the complete conceptual model.

A variety of analyses for conceptual clustering, and visualization of the domain can be completed using the distance-based ratings in the grid (Shaw, 1980; Shaw & Gaines, 1986). These will aid the expert in reviewing and defining the domain structure in his conceptual model. At a more formal stage of analysis classes, objects etc. can be produced using empirical induction.

Empirical Induction is basically a method that can be applied to a set of rules (or equivalent) from a set of cases (Forsyth & Rada, 1986). It is necessary to have a training set of critical (or other — see Gaines, 1989) cases which encompass crucial and complete infomation. The cases are distinguished by a set of attributes, in a similar way to the repertory grid attributes. The decision of the expert in each of the cases is also needed and an induction algorithm such as ID3 (Quinlan, 1987). ID3, or Interactive Dichotomizer 3, works the best if the data is not corrupted by 'noise'. A subset of a certain size (window) is selected from the training set and an algorithm applied to derive a rule for this window. All exceptions to this rule are noted and the algorithm re-applied with these exceptions until none are found. When the whole set of rules are found they are formed into a decision tree.

Induce (Dietterich & Michalsky, 1981) is another induction algoritmn based on earlier experience with a system called AQ11. A search technique is used to explore the set of examples and generalize very specific rules formed from an initial subset of the training set. There are many other induction systems notable Prism (Cendrowska, 1987) and Induct (Gaines, 1989). In general, the quality of the induced results depends on the quality of the initial training set which must be drawn up by an expert. The resulting rules are usually tested on a quite separate set of examples than the training set, and the final set of rules discussed with the expert (Hart, 1987).

Gaines (1989) reports a study of the tradeoff between the knowledge needed for expertise transfer and the cases required for the induction training set. The results indicate that a mixture of correct rules, adequate rules, a critical set of cases described in terms of a set of relevant (or including some irrelevant) attributes, and even some incorrect decisions can produce the same minimal set of correct decision rules. In other words, the lack of a first class expert who can abstract the essential knowledge from his model, and represent it clearly in an intermediate knowledge base can be offset by the analysis process which produces the computer knowledge base. There is scope for further exploration of this insight into the relationship between empirical induction and expertise transfer.

CONCLUSION

In this paper we have identified one of the major problems in knowledge acquisition as the lack of an overall methodological framework which logically supports the process of knowledge acquisition, identifies the characteristics of the products of knowledge acquisition, and offers a set of ordered assumptions about the nature of knowledge and its characteristics. We have shown the modeling processes involved from the perspective of the expert trying to communicate his view of the target system and transfer it into computer implementable form. We have identified the distinct proccesses of elicitation, analysis and implementation, but have not gone beyond this stage of development of a knowledge-based system. Finally, we have identified the separate knowledge representations of the intermediate knowledge bases which can be used to help the expert review and refine his conceptual model, and the computer knowledge bases which may be unrecognizable by the expert as related to his developing models.

Many manual and automatic elicitation methods exist. Some are well-founded in basic underlying theories from established disciplines, but the majority are based on *ad hoc* methods that seem to work. What has to be established by the advocates of these various methods is an underlying justification, not only from another discipline, but also from a well-established and tested discipline such as psychology, linguistics, philosophy or systems theory. Any less well-established theories need, in turn, to be grounded in a basic discipline.

Shaw and Gaines (1987) list requirements for a computer-based knowledge support system which will support the expert in his or her endeavours to externalize problem-solving behaviors:

1. The knowledge support system should be domain independent;
2. The knowledge support system should be directly applicable by experts without intermediaries;
3. The knowledge support system should be able to access a diversity of knowledge sources including text, interviews with experts, and observations of expert behavior;
4. The knowledge support system system should be able to encompass a diversity of perspectives including partial or contradictory input from different experts;
5. The knowledge support system should be able to encompass a diversity of forms of knowledge and relationships between knowledge;
6. The knowledge support system should be able to present knowledge from a diversity of sources with clarity as to its derivation, consequences and structural relations;
7. Users of the knowledge support system should be able to apply the knowledge in a variety of familiar domains and freely experiment with its implications;
8. The knowledge support system should make provision for validation studies;
9. As much of the operation of the knowledge support system as possible should be founded on well-developed and explicit theories of knowledge acquisition, elicitation and representation;
10. As the overall knowledge support system develops it should converge to an integrated system.

These requirements provide a first checklist for developers, users and supporters of existing systems and techniques. In addition, we would include support of the analysis phase of

knowledge acquisition, in such a way that the conceptual model of the expert is preserved as much as possible, and the analysis results in knowledge representation schemes that can be adequately applied in computerized knowledge bases. The above requirements are a start in aiding people to assess more completely the validity or otherwise of assumptions made about the type of knowledge which is being elicited by the various tools and techniques. Clearly, they must be designed to support the expert as much as possible and to help him or her in preserving an existing series of models that lead to effective interaction with the target system.

ACKNOWLEDGEMENTS

Financial assistance for this work has been made available by the Natural Sciences and Engineering Research Council of Canada. We are grateful to many colleagues for discussions which have highlighted many of the issues raised in this paper, and in particular to Brian Gaines for his critical comments and stimulating discussions.

REFERENCES

Addis, T. (1989). The science of knowledge: a research program for knowledge engineering. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89)**. Paris, France (June).

Anderson, J.R. (1983) **The Architecture of Cognition.** Harvard University Press: Cambridge, Mass.

Bainbridge, L. (1986). Asking questions and accessing knowledge. **Future Computing Systems** 1(2), 143-149.

Belkin, N.J., Brooks, H.M. & Daniels, P.J. (1986) Knowledge elicitation using discourse analysis. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the First AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.3-0-3-16. Banff (November).

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.

Brachman, R.J. (1979) On the epistomological status of semantic networks. N.V. Findler (Ed) **Associative Networks,** Academic Press: New York..

Bresnan, J. (Ed) (1982) **The Mental Representation of Grammatical Relations.** MIT Press: Cambridge, Mass.

Breuker, J. & Wielinga, B. (1987) Use of models in the interpretation of verbal data. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum.

Cendrowska, J. (1987) An algorithm for inducing modular rules. **International Journal of Man-Machine Studies 27** (4), 349-370 (October).

Chandrasekaran, B (1988) Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. **The Knowledge Engineering Review, 3**(3), 183-211.

Chomsky, N. (1975) **Logical Structure of Linguistic Theory.** New York: Plenum.

Churchman, C.W. (1971) **Design of Inquiring Systems.** New York: Basic Books.

Clancey, W. (1989) Viewing knowledge bases as qualitative models. **IEEE Expert,** 4(2), 9-23.

Compton, P. & Jansen, R. (1989). A philosophical basis for knowledge acquisition. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89)**. Paris, France (June).

Craik, K.J.W. (1943) **The Nature of Explanation.** Cambridge University Press.

Dietterich, T. & Michalsky, R. (1981). Inductive Learning of Structural Descriptions, **Artificial Intelligence, 16.**

Ericsson, K.A. & Simon, H.A. (1984). **Protocol Analysis: Verbal Reports as Data**. Cambridge, Mass. MIT Press.

Fodor, J.A. (1980) **Representations.** Cambridge Mass.: MIT press.

Forsyth, R. & Rada, R. (1986). **Machine Learning Applications in Expert Systems and Information Retrieval,** Ellis Horwood: Chichester.

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. (1989) An Ounce of Knowledge is Worth a Ton of Data: Quantitative Studies of the Trade-Off between Expertise and Data based on Statistically Well-Founded Empirical Induction. **Proceedings of 6th International Workshop on Machine Learning,** pp.156-159. San Mateo, California: Morgan Kaufmann (June).

Gaines, B.R. & Shaw, M.L.G. (1987). Knowledge support systems. **ACM MCC-University Research Symposium.** Austin, Texas: MCC. pp.47-66.

Gazdar G. (1981). On syntactic categories. **Philosophical Transactions of the Royal Society of London,** Series B, **295,** pp 267-83.

Graesser A.C. & Clark, L.F. (1985). **Structures and Procedures of Implicit Knowledge.** New Jersey: Ablex.

Gilhooly K.J. (1987). Mental modeling: a framework for the study of thinking. D.N. Perkins, J. Lochhead & J.C. Bishop (Eds) **Thinking: the Second International Conference.** NJ: Erlbaum.

Harrison, A. & Bramson, (1982) R.M. **Styles of Thinking.** New York: Doubleday.

Hart, A. (1987). Role of induction in knowledge elicitation. A.L. Kidd (Ed) **Knowledge Acquisition for Expert Systems: A practical handbook.** New York: Plenum.

Hayward, S.A., Wielinga, B.J. & Breuker, J.A. (1986) Structured analysis of knowledge. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the First AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.18-0-18-6. Banff.

Hirst, G. & Regoczei, S. (1989). On 'extracting knowledge from text': modelling the architecture of language users. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89).** Paris, France (June).

Johnson L. & Johnson, N.E. (1987) Knowledge elicitation involving teachback interviewing. Kidd, A. (Ed) **Knowledge Elicitation for Expert Systems: A Practical Handbook.** Plenum Press.

Johnson, P.E. (1986) Cognitive models of expertise. **Symposium on Expert Systems and Auditor Judgment.** University of Southern California.

Johnson, P.E., Zualkernan & Garber, S. (1987). Specification of expertise: **International Journal of Man-Machine Studies, 27**(2), 161-181 (February).

Johnson-Laird, P.N. (1983) **Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness.** Cambridge University Press.

LaFrance, M. (1987) The knowledge acquisition grid: a method for training knowledge engineers. **International Journal of Man-Machine Studies, 27**(2), 245-255 (February).

Gentner, D., & Gentner, A.L. (1983) Flowing waters or teeming crowds: mental models of electricity. D. Gentner & A. Stevens, (Eds.), **Mental Models.** Hillsdale, N.J.: Erlbaum.

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

McDermott, J. (1988) Preliminary steps toward a taxonomy of problem solving methods. Marcus, S. (Ed) **Automating Knowledge Acquisition for Expert Systems,** Kluwer: Boston.

McGraw, K.L., & Harbison-Briggs, K. (1989). **Knowledge Acquisition Principles and Guidelines,** Prentice Hall: New Jersey.

Minsky, M. (1975) A framework for representing knowledge. P.H. Winston (Ed.), **The Psychology of Computer Vision.** New York: McGraw Hill.

Norman, D.A. (1983) Some observations on mental models. D. Gentner & A. Stevens, (Eds.), **Mental Models.** Hillsdale, N.J.: Erlbaum.

Nystedt, L. (1983) The situation: a constructivist approach. J.A. Adams-Webber & J.C. Mancuso (Eds) **Applications of Personal Construct Theory.** New York: Academic Press.

Paivio, A.U. (1977) Images, propositions and knowledge, J.M. Nicholas, (Ed.) **Images, Perception and Knowledge.** Reidel.

Pask, G. (1975). **Conversation, Cognition and Learning,** Elsevier, Amsterdam.
Pylyshyn, Z.W. (1984) **Computation and Cognition.** Cambridge, Mass.: MIT Press.
Quinlan, J.R. (1987) Simplifying decision trees. **International Journal of Man-Machine Studies 27** (3), 221-234 (September).
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. (1980) **On Becoming a Personal Scientist.** London: Academic Press.
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. (1987) KITTEN: Knowledge Initiation and Transfer Tools for Experts and Novices. **International Journal of Man-Machine Studies 27**(3), 251-280.
Sloman, A. (1980) **The Computer Revolution in Philosophy,** Harvester.
Twine, S. (1989). A model for the knowledge analysis process. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89).** Paris, France (June).
Welbank, M. (1987) Knowledge acquisition update: Insight study #5. **Systems Designers,** Pembroke Broadway, Camberley, Surrey, UK.
Winston, P.H. (1984) **Artificial Intelligence.** (2nd ed). Reading. Prentice-Hall.
Woodward, B. (1988) Knowledge engineering at the front-end: defining the domain. Boose, J.H. & Gaines, B.R. (Eds) **Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop.** pp.37-1-37-25. Banff (November).
Woodward, B. (1989) Integrating task demands and problem-solving methods to develop useful taxonomies for knowledge acquisition. **Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW'89).** Paris, France (June).

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Design Knowledge Capture and Alternatives Generation Using Possibility Tables in *Canard*[1]

David B. Shema, Jeffrey M. Bradshaw, Stanley P. Covington, and John H. Boose
Advanced Technology Center, Boeing Computer Services
P.O. Box 24346, M/S 7L-64, Seattle, WA 98124-0346, (206) 865-3404, dshema@atc.boeing.com

ABSTRACT

During the evolution of a design concept, designers must integrate diverse sources and kinds of information about requirements, constraints, and tradeoffs. In doing so, they make certain assumptions and develop criteria against which alternatives are evaluated for suitability. Unfortunately, much of this process is implicit, making later review difficult if not impossible. When requirements change, impacts on the design are difficult to trace. This can lead to costly rework or serious errors. We are developing *Canard*, an automated tool which uses possibility tables, constraints, and knowledge bases to assist in the generation of design alternatives consistent with goals and constraints. The facility also attempts to capture and document assumptions and tradeoffs made during the design process. We present an example which illustrates the use of *Canard* for a simple configuration problem. A more complex example traces the activity of a Boeing expert building a possibility table for robot arm design. Finally, the application of *Canard* to a NASA corporate memory facility project is described.

1. INTRODUCTION

1.1. Problems of Design Knowledge Capture and Alternatives Generation

During the evolution of a design concept, designers must integrate diverse sources and kinds of information about requirements, constraints, and tradeoffs. In doing so, they make certain assumptions and develop criteria against which alternatives are evaluated for suitability. Unfortunately, much of this process is implicit, making later review difficult if not impossible. When requirements change, impacts on the design are difficult to trace. This can lead to costly rework or serious errors.

An automated tool provides at least documentation of the design history. The information used in defining the alternatives can be captured and stored in a retrievable manner. With such a capability, a more complete record of the design is available for later review and revision. The knowledge and experience of the designer is also available to others who may be facing a new problem that is similar to the old one.

In addition to preservation of the design history, we are concerned with helping designers better explore the space of alternatives. Cognitive scientists have long known that people typically retrieve only a small fraction of available alternatives when generating hypotheses (Wise, 1985). People tend to anchor on initial guesses, giving insufficient regard to subsequent data. For various other reasons, people may not be able to visualize whole classes of possibilities (Kahneman, Slovic, and Tversky, 1982).

---

[1]The work described in this paper was supported by NASA contract NAS2-12108.

Although it would be impossible in practice to guarantee that all relevant alternatives were considered, an automated tool could help designers develop a richer set of alternatives. By coupling the alternative generation facility to analysis tools, the system could also help designers identify alternatives that differ slightly from program requirements. The designer would get a better feel for the effects of the constraints on different alternatives and could better evaluate the consequences of assumptions and tradeoffs.

1.2. Approach to Building Tools for Automated Support

The use of knowledge-based systems has been proposed as a way to both preserve a more complete record of available alternatives (i.e., capturing design knowledge) and to help persons explore feasible combinations of design parameters that might otherwise go unconsidered. To this end, we are developing knowledge-based tools that can work in conjunction with an automated possibility table facility.

1.2.1. Analysis and Synthesis Problems

There is a traditional distinction in the literature between analysis and synthesis problems (Rubinstein, 1975; Wise, 1985). *Analysis* problems are those in which the alternatives can be conveniently enumerated (e.g., classification, diagnosis, prediction), while *synthesis* problems are those where the main task is constructing feasible alternatives in a manner that is consistent with hard constraints and with soft (optimal or "good enough") constraints with respect to objectives (e.g., design, planning, configuration, scheduling). As Clancey (1984) observes, however, real-world problems do not always fall neatly into one of these two categories; generally the solution to complex problems involves the integration of several reasoning strategies.

A number of approaches have evolved to the solution of synthesis problems. More traditional approaches, from fields such as operations research, include optimization techniques (e.g., linear, nonlinear, and dynamic programming) and modeling of dynamic systems (e.g., control theory, simulation). Knowledge-based approaches have largely bypassed these methods in favor of various heuristic constraint-satisfaction techniques. These approaches sometimes employ a variation of an incremental "propose-and-revise" strategy with the aim of satisficing rather than optimizing a solution (e.g., Marcus, 1987).

Traditional and knowledge-based approaches to solving synthesis problems are similar in many respects. Both usually begin with a statement of desired outcome states, in the form of constraints, and attempt to synthesize feasible or acceptable alternatives through exploration of the bounds and the interactions of the constraints. Since there will likely be several acceptable alternatives, the synthesis phase will be followed by analysis to determine which alternative best meets objectives such as least cost, maximum reliability, and so forth. Approaches to such problems generally do not differ with respect to these general steps, but rather in what specific methods are applied to generate hypotheses and reduce the extent of the search for a good solution. We describe our view of the design process more completely in Bradshaw, Boose, Covington, and Russo (1989).

A tool known as a *possibility table* provides an integrating interface for the synthesis and analysis of design alternatives. Within this facility, components, constraints, and objectives can be synthesized into design alternatives with automated assistance. These alternatives are analyzed, using additional tools available within the interface, to make the consequences of a particular choice explicit and available for review at any time. We describe these facilities below.

1.2.2. Possibility Tables

The possibility table representation is not a new idea. Manually developed strategy tables have been used by decision analysts as one way of generating and representing complex alternatives (McNamee and Celona, 1987). Related approaches, such as Zwicky's morphological charts, (Zwicky, 1969) have been manually employed by designers for many years.

Possibility tables are used to structure information rabout complex alternatives, outcomes, or plans[1] (Figure 1). When used for a design or configuration problem, columns in the possibility table typically represent essential components or functions that compose the artifact being designed. Within each column, the various possibilities listed identify alternatives for that component. To construct a design alternative, a designer selects a possibility from each column, defining a unique path through the table. The names of these alternatives appear in the leftmost column.

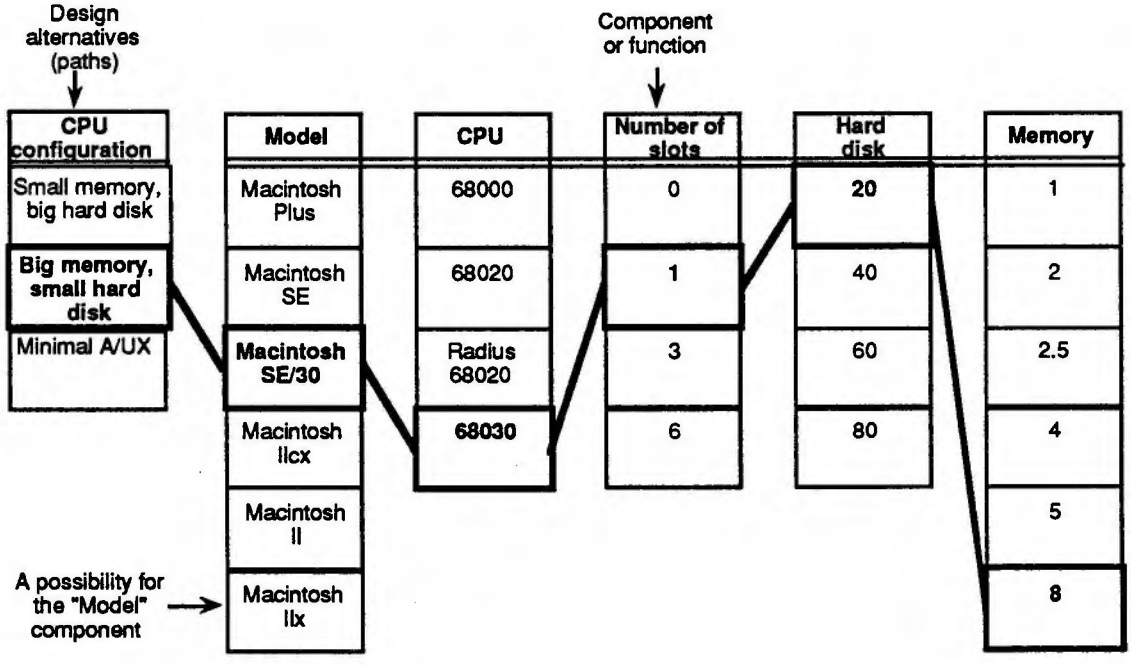


Figure 1. A possibility table for configuration of a Macintosh computer system.

[1] *Possibility table* is the generic term referring to the graphical representation. We sometimes use the more specific terms *strategy tables* to refer to tables defining alternatives, *outcome tables* to refer to tables defining outcomes, and *agenda tables* to refer to tables defining portions of a plan.

1.2.3. *Canard:* A Tool That Integrates Possibility Tables With Knowledge-based Tools in Aquinas and Axotl

Canard automates the possibility table representation and extends its logic and structure to allow knowledge-based inference and the representation of more complex problems than could be handled by manual approaches such as hierarchical tables or explicit representation of constraints. *Canard* receives assistance in alternatives generation and design knowledge capture through its integration with two additional tools: Aquinas, an automated knowledge acquisition workbench, and Axotl, a knowledge-based decision analysis and process management workbench (Figure 2).

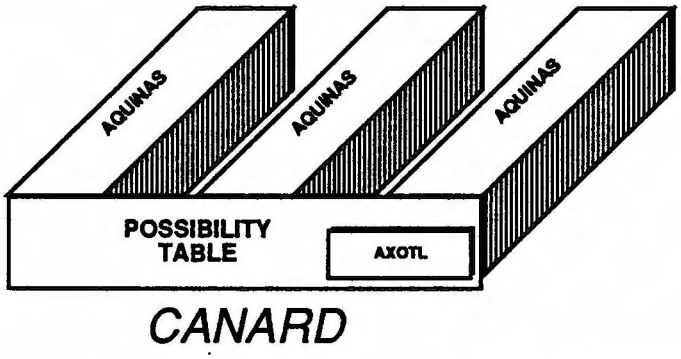


Figure 2. *Canard* is a design tool that integrates possibility tables with knowledge-based tools in Aquinas and Axotl.

Aquinas, an expanded version of the Expertise Transfer System (ETS) (Boose, 1984, 1985, 1986), is a workbench that supports several knowledge acquisition activities. The system interviews experts and helps them analyze, test, and refine a knowledge base. It uses methods from personal construct theory, an approach that grew out of George Kelly's research and experience as a clinical psychologist (Kelly, 1955). Activities supported by Aquinas include eliciting distinctions, decomposing problems, incrementally testing knowledge bases, integrating data types, automatically expanding and refining the knowledge base, using multiple sources of knowledge, and using constraints in the inference process. In the context of *Canard,* the tools in Aquinas are used to elicit and structure information about design component possibilities and gather the constraints and objectives that guide a designer in the selection of these possibilities. Analysis tools in Aquinas help designers determine the adequacy of the constraints and objectives and focus their attention on descriptions needing further refinement. Aquinas is discussed in greater detail in Boose and Bradshaw (1987) and Boose, Shema and Bradshaw (1988).

Axotl combines a decision analysis workbench with knowledge-based tools. The decision analysis workbench contains a graphical influence diagram editor (Howard and Matheson, 1980), which is used for creating and refining models of alternatives, preferences, and uncertainties relevant to a specific decision. The knowledge base tools in Axotl can be configured with application-independent knowledge (i.e., knowledge of decision analysis tools and methodology) and application-specific knowledge (i.e., knowledge about a particular domain) to provide guidance and help during a consultation. Within the *Canard* interface, the knowledge base tools in Axotl can be used to graphically represent and reason about constraints on a design and to represent knowledge about the design process itself. Additionally, the decision analysis workbench may be useful for decisions involving significant uncertainties, complex tradeoffs, or high stakes. *Axotl* is described in more detail in Bradshaw and Boose (1989) and Bradshaw, Covington, Russo, and Boose (1989).

Canard enhances possibility tables with constraint-handling tools which reduce the possible solution space and also capture important information about the design. *Canard* allows both hard and soft constraints. A designer adds hard constraints to possibilities to capture information about

incompatibilities and interdependencies between component possibilities. During alternatives generation, these hard constraints prevent any incompatible components from being selected.

Soft constraints and utility scores may be added to attributes which characterize generated paths. The desired values of these attributes are the design goals. Conflicts between two or more design goals often necessitate making tradeoffs between the goals. For example, a goal of low cost is often in conflict with a goal of high reliability. Using *Canard,* the designer can specify the acceptable ranges of an attribute and map attribute values to the utility of these values. The system could then guide the designer toward possibilities that optimized the tradeoffs between the soft constraints.

Possibilities may be selected manually by the designer in an exploratory mode. However, for large problems it would be impractical to force designers to assign each possibility directly. In these cases, an iterative search proposes new alternatives based on permutations of the constraint space to assist in generating alternatives. Through a similar procedure, the system can hypothesize new constraints based on examples of previously defined alternatives. By keeping track of what has been tried before, the tool assists the designer in covering the possible solution space.

Possibility tables in *Canard* have a graphical interface which facilitates creation and modification of design information. The designer creates new components and possibilities and defines alternative paths using the mouse. The interface also assists the designer in managing the complexity of large design problems by presenting the design in an easily comprehensible view. Such an interface integrates well with the environments of *Aquinas* and *Axotl.*

The graphical possibility table facility in *Canard* is being implemented in ParcPlace Smalltalk-80 within the DDUCKS environment (Bradshaw, Covington, Russo, and Boose, 1989). DDUCKS supports coordination of conventional and knowledge-based applications running concurrently under MultiFinder on the Macintosh or across a network. This facility allows the possibility table tools in *Canard* to have access to all the functionality of *Aquinas* and *Axotl.*

2. EXAMPLE APPLICATIONS OF *CANARD*

To illustrate the use of *Canard,* we will describe a hardware configuration decision made recently as part of planning for a new project. Following this example, we present a more complex possibility table developed by a Boeing robotics expert. Finally, we will discuss how possibility tables will be used in more complex problems as part of a NASA corporate memory facility (CMF) study.

2.1. Personal Computer Configuration

We recently began a new project which required the purchase of two or three Macintosh computer systems. Specific hardware requirements had to be met while remaining within a $15,000 budget. In addition, from the long-term perspective, team members were interested in configuring a system that would be expandable and flexible for future projects.

We will present the example in terms of seven different kinds of activities supported by *Canard.* The activities are highly interrelated, and it would be unreasonable to suppose that they would be followed in any sort of strict order. At any time, the designer might move from one activity to another according to the type of knowledge being entered into *Canard.* The designer might add or modify components, possibilities, and constraints at any time. The activities, described below, are:

1. Specifying components and possibilities.

2. Specifying inter-component compatibility constraints.
3. Specifying intra-component attributes, preferences, and constraints.
4. Specifying inter-component attributes, preferences, and constraints.
5. Proposing alternatives by specifying possibility paths.
6. Automating path generation.
7. Automating constraint generation.

2.1.1. Specifying Components and Possibilities

The designer first needs to build the basic structure of the possibility table. The initial components of the computer system are identified and become the column headers (Figure 3). Possibilities are listed under each component. The leftmost column is reserved for the names of the alternatives that will be generated.

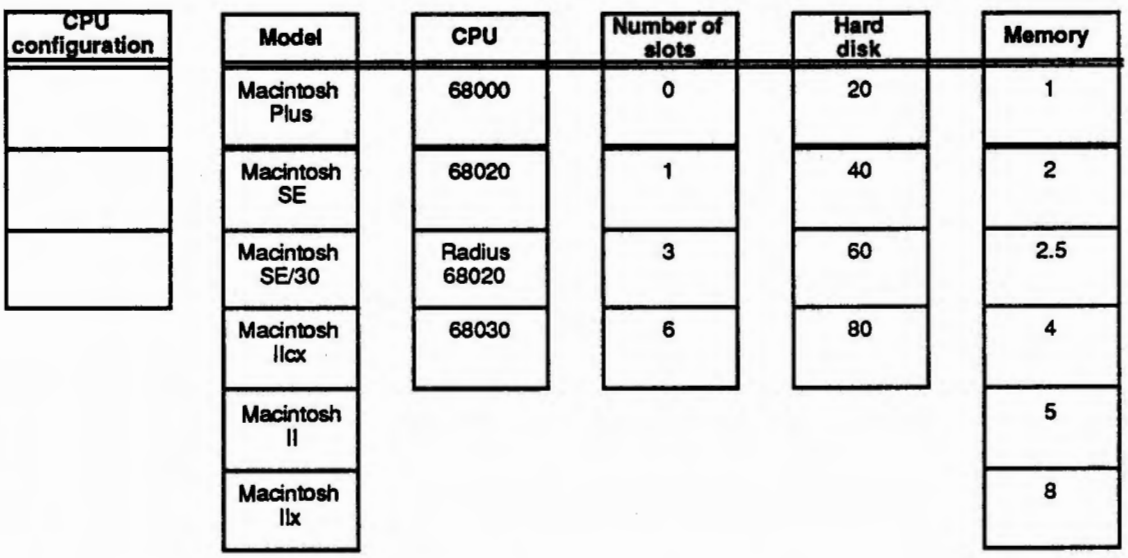


Figure 3. Components and possibilities are specified in the possibility table.

2.1.2. Specifying Inter-component Compatibility Constraints

Hard constraints are added to the table to capture information about incompatibilities between combinations of component possibilities. Compatibilities are stored in a tree-like structure; using tools from Axotl, the designer prunes the tree to remove the incompatibilities. Such a representation permits specification of complex incompatibilities between three or more interacting possibilities, as well as simple incompatibilities between pairs of possibilities. Simple incompatibilities may also be specified directly in the possibility table using a point-and-click mechanism. The designer selects combinations of incompatible possibilities, and *Canard* automatically prunes the compatibility tree.

A partial compatibility tree, showing the CPU, memory, and hard disk configurations available for the Macintosh SE, is shown in Figure 4. The single hard disk branch is unlabeled and unexpanded, indicating that all hard disk possibilities are compatible with every other type of possibility shown. The constraints specified in the compatibility tree cannot be relaxed.

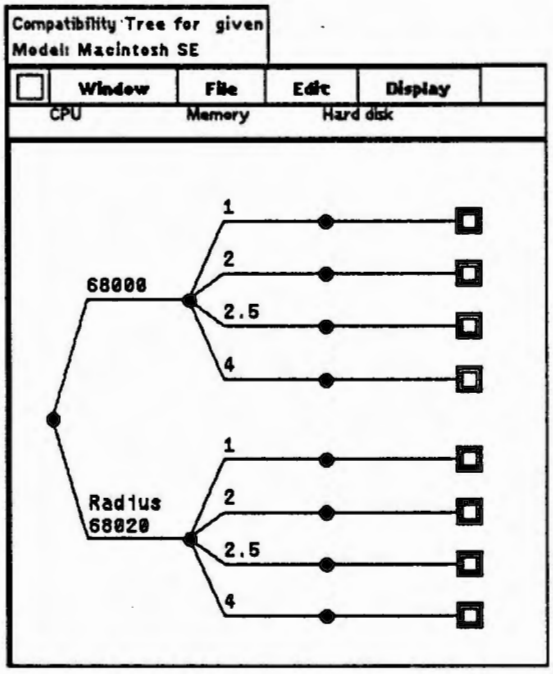


Figure 4. A partial compatibility tree, showing the CPU, memory, and hard disk configurations available for the Macintosh SE.

2.1.3. Specifying Intra-component Attributes, Preferences, and Constraints

The designer creates an Aquinas knowledge base for each component. Within Aquinas, a number of useful representation, interviewing, and analysis techniques have been developed to help designers determine distinguishing features of component possibilities. Many of these techniques are based on the research of George Kelly (1955), a clinical psychologist who emphasized the foundational role of distinctions (*personal constructs*) underlying the processes of perception and reasoning. For example, using a *triadic elicitation* interviewing technique, Aquinas asks designers to define similarities and differences by considering groups of component possibilities presented three at a time: "Think of an important attribute that two of Macintosh Plus, Macintosh SE, and Macintosh SE/30 share, but that the other does not. What is that characteristic?" After giving Cost (More Money/Less Money) as that characteristic, the person might then be asked about Macintosh SE, Macintosh SE/30, and Macintosh IIcx and come up with a second characteristic Size (Small-Medium-Large), and so forth.

One way of representing this information is through a *repertory grid* (Figure 5), a matrix with elements (i.e., component possibilities) ranged along the bottom and constructs (i.e., dimensions of similarity and difference between elements), defined by extension, as a horizontal row of values (or probability distributions) within the matrix. The grid presentation allows the user to see patterns of similarity and difference that would otherwise be difficult to grasp. Analysis techniques in Aquinas (e.g., similarity analysis, cluster analysis) exploit these patterns to help users discriminate more carefully among similar concepts as part of model refinement. Implication analysis helps users discover important dependencies between constructs.

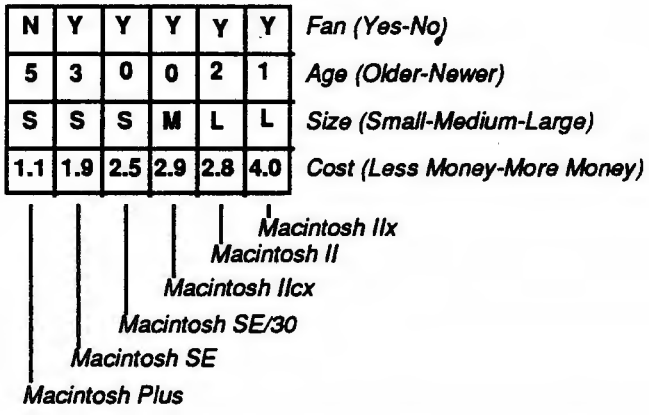


Figure 5. Model possibilities are compared and contrasted along attributes defined using *Aquinas*.

The grid can be viewed as being "plugged in" to the back of a column of the possibility table (Figure 6) and accessible to the possibility table. The attributes and possibilities for a particular column "slice" form the rows and columns of the repertory grid, and the values for each possibility with respect to each attribute constitute the grid ratings.

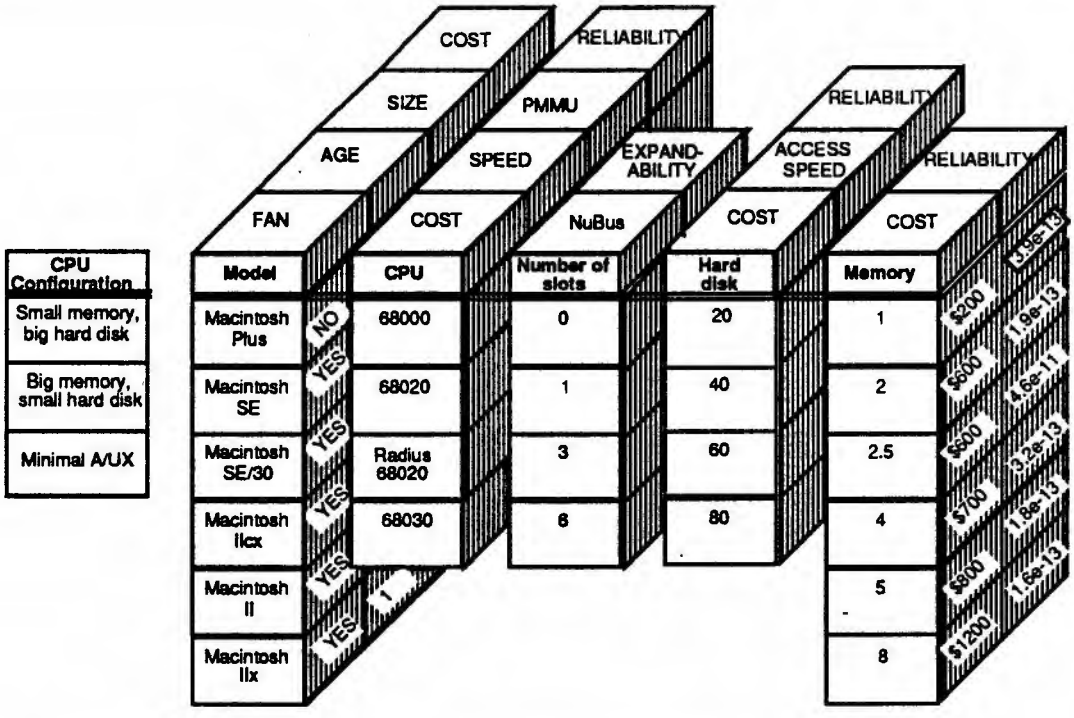


Figure 6. Aquinas knowledge bases "plug into" the back of the possibility table.

The knowledge base is used during the design process to guide the designer in selecting the best possibilities for each component. Hard constraints filter out incompatible possibilities. Tradeoffs are computed from grid information, given each attribute value for each component possibility and the preferred attribute values (soft constraints) for each component, resulting in overall utility scores for each possibility.

Constraints and preferences for each attribute can be entered using an interactive plotting facility. Designers use the mouse to assign relative utility to the range of values for an attribute (Figure 7). Utilities between 1 and 100 are used as soft constraints in the selection of the best possibility for each component. If a particular possibility has a rating that maps to a utility of 0, *Aquinas* treats the information as a hard constraint and eliminates the possibility from further consideration. For example, in Figure 7 the designer has asserted that any model with a base price of greater than $3,500 has a utility of zero. This eliminates the Macintosh IIx, with a cost of $4000, from further consideration (see Figure 5). In the possibility table, the Macintosh IIx entry will be inverted to signify that it will be excluded from further consideration (see Figure 9). Constraints can also be entered using a constraint language.

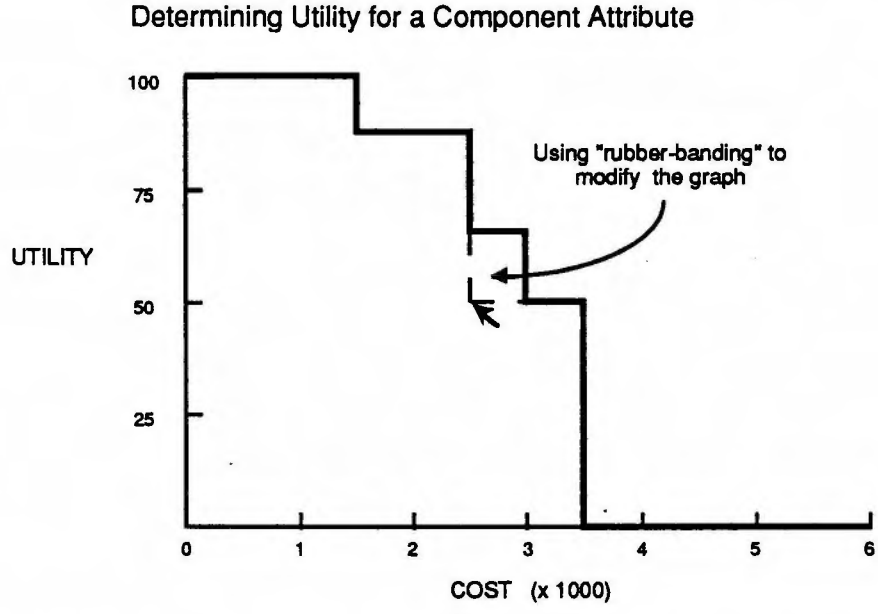


Figure 7. Graphical facility allows attribute values to be mapped to a scale of utility.

Using Aquinas, relative weights are assigned to the attributes and used to compute an overall utility score for each possibility. As information is changed in the grid, the utility score can be dynamically computed and displayed next to the possibilities in the possibility table. This score will guide designers in selecting the best possibility within a column, given previous selections, when they construct paths through the table.

2.1.4. Specifying Inter-component Attributes, Preferences, and Constraints

Inter-component (or path) attributes, preferences, and constraints characterize the possibility paths that define each alternative. The attributes are important dimensions shared across some subset of the components. Preferences on these inter-component attributes specify the overall goals which should be optimized for the design. (A utility metric measures the degree to which an attribute value satisfies a preference – the higher the utility, the better the alternative.)

The designer first provides the name, scale type (nominal, ordinal, interval, or ratio), range, interval, and weight information for each attribute. Next, the designer defines a function that will calculate the inter-component attribute value. During the specification of a path, the function will dynamically update the total value of the path attribute as the possibilities are selected. The current value is displayed in an active gauge (see Figure 9).

To see how well the alternative satisfies overall design goals, designers map inter-component attribute values to a utility scale in the same manner as for intra-component attributes (Figure 8; compare to Figure 7).

In Figure 8, the designer has defined the path attribute TOTAL.COST. The range of TOTAL.COST is 0 to 7000 in intervals of 500. The weight of 1.0 specifies the relative contribution of TOTAL.COST to the overall path utility score in comparison with other path attributes. On the utility graph, the designer has indicated that costs between 0 and 4500 are equally preferable. Costs between 4500 and 6000 are acceptable, but not nearly as desirable. A cost greater than 6000 has zero utility: the designer wants to avoid such a cost, if possible.

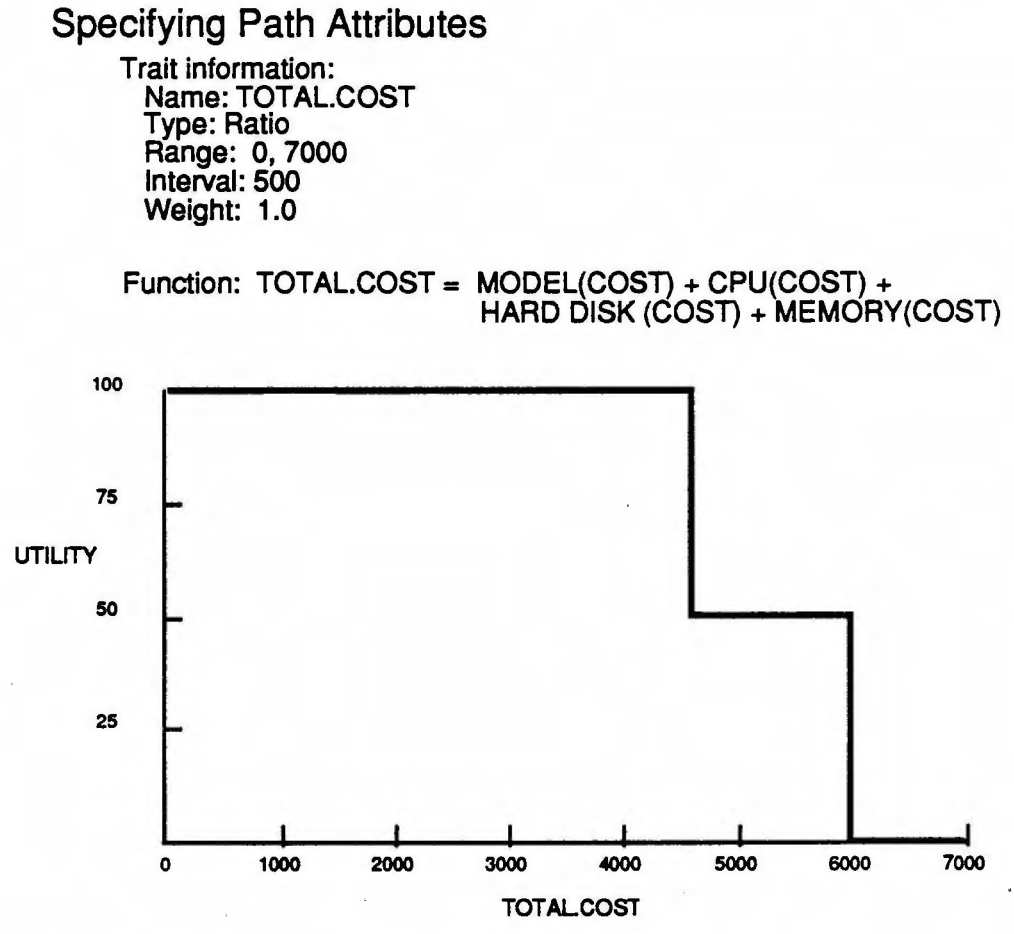


Figure 8. The designer provides information about the TOTAL.COST attribute.

2.1.5. Proposing Alternatives by Specifying Possibility Paths

An alternative may be generated by selecting a possibility from each component using the mouse. As each possibility is selected, incompatible possibilities in other components are inverted to signal their incompatibility with hard constraints on the possibilities already selected. Graying of possibilities signifies level of desirability with respect to soft constraints. Active gauges associated with the path attributes display the current values of their function and a utility score. If a value of a path attribute exceeds the limit specified in a hard constraint, the designer may need to retract one or more possibility selections.

In Figure 9, the designer is in the process specifying a path for the alternative "*Small memory, big hard disk* " through the possibility table. *Model*, *Slot*, and *Disk* components have already been selected and are outlined. The inverted possibilities indicate that the possibilities that were found to be incompatible when the *Model* possibility was selected. The *Model* possibility *Macintosh IIx* was eliminated prior to path specification because of the hard constraint on CPU cost. The table and active gauges associated with the path attributes TOTAL COST and AVE RELIABILITY display the values of the cost and reliability functions as well as their utility score.

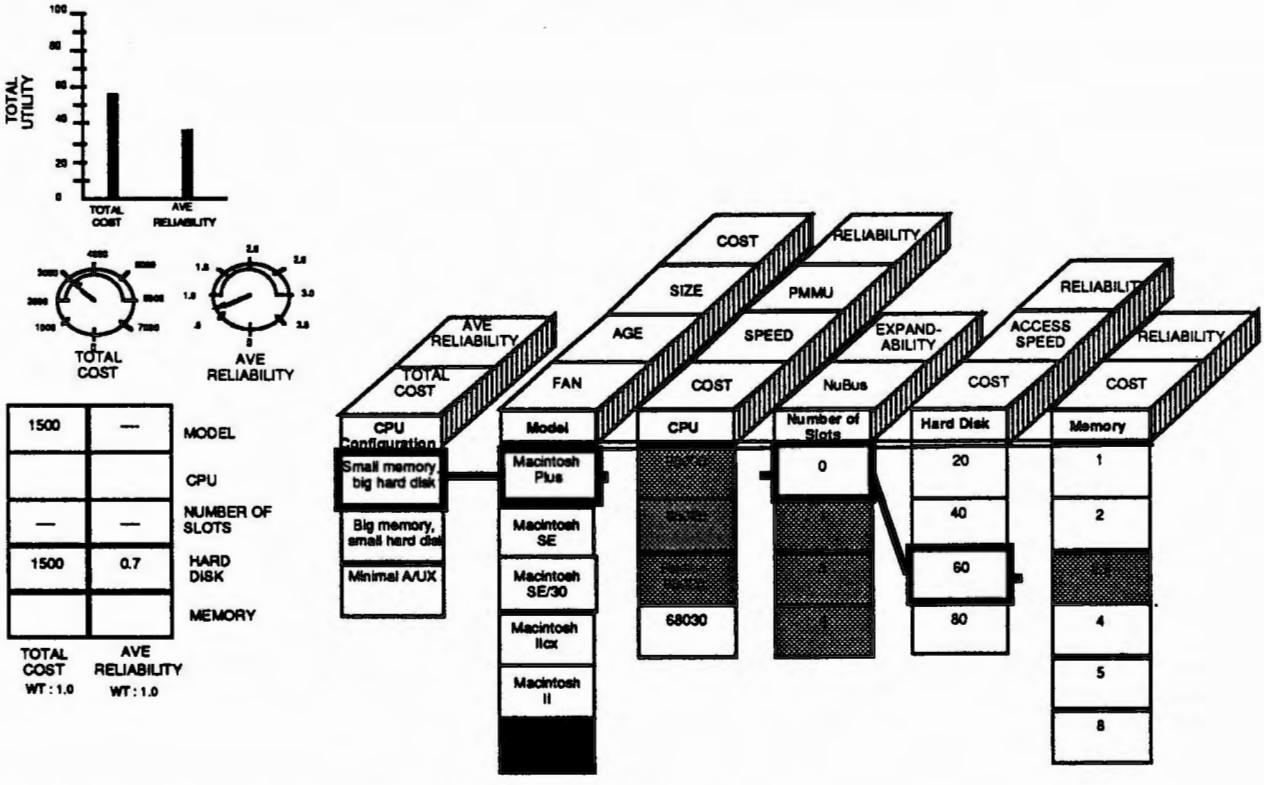


Figure 9. Specifying a possibility path to define an alternative.

Labeling the specified path is in and of itself an interesting part of the alternatives generation process. The designer may supply an initial label that describes the design concept being defined and then modify the description as the path is specified. As the designer experiments with different possibilities, naming the concepts embodied in the new paths helps the designer gain additional insights about the problem.

2.1.6. Automating Path Generation

Knowledge of the attributes, constraints, and preferences of components and possibilities permits *Canard* to automatically propose optimal paths through the table. It examines existing constraints and preferences and generates new paths that avoid component incompatibilities and maximize the combined utility of inter-component and intra-component attributes. Through this process, designers may be led to explore alternatives that might have otherwise gone unconsidered. The automatically generated path is presented to the designer, who may accept it or reject it. If it is accepted, the new alternative can be labeled and considered in more detail. If a proposed path is rejected, the designer is asked to state a rationale for rejection in the form of new or modified constraints.

2.1.7. Automating Constraint Generation

When a number of good alternative paths have been specified, *Canard* can analyze existing paths and query the designer about combinations of component possibilities that have not been explored. If the designer can explain why the combination has not been explored previously, a new constraint has been discovered. Otherwise, a new alternative can be proposed.

2.2. Robot Design

An engineer at Boeing with a background in robotics was asked to create a possibility table that could assist with the design of a robot arm for use on a vehicle such as Space Station Freedom or the Space Shuttle. Because *Canard* was not fully implemented at the time, the possibility table was constructed manually. We include the example to illustrate what *Canard* shall be capable of. The engineer first named three of the components used to construct an arm (*Internal Power, End Effector, Arm Base Type*) and supplied possibilities for each component. Asked to differentiate between the possibilities for *Arm Base Type,* he identified two attributes: Mobility and Rigidity. (Figure 10.)

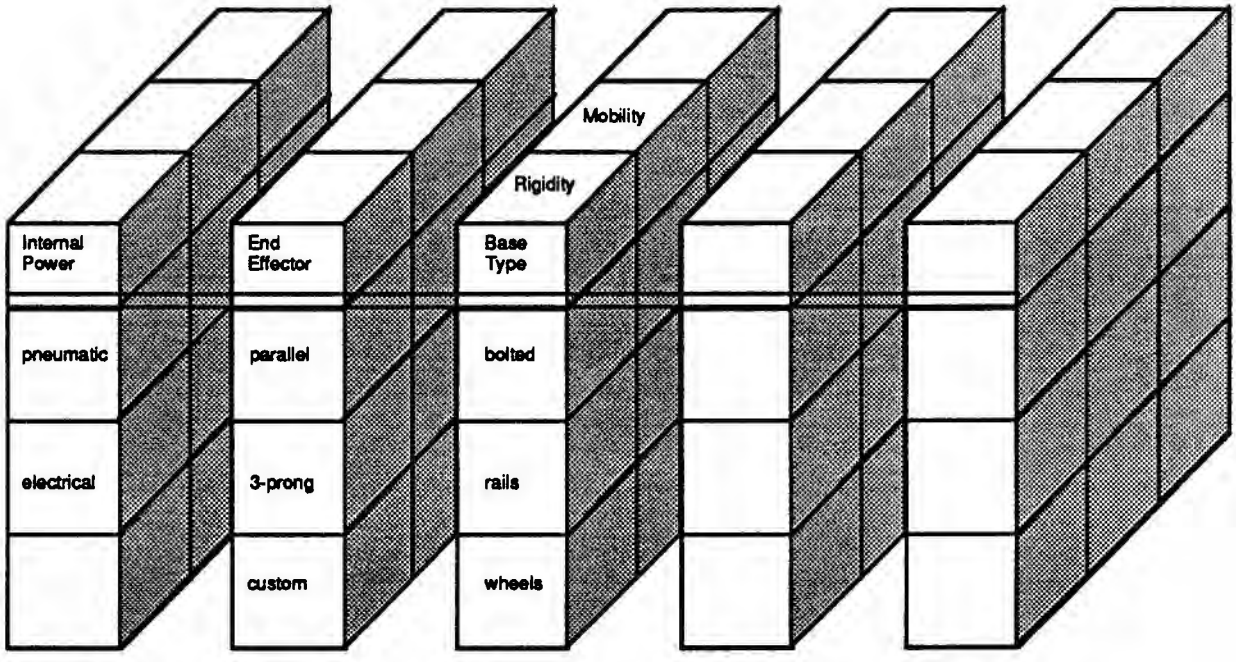


Figure 10. Robot arm possibility table under construction

The engineer supplied additional components for the construction of a robot arm and gave a few possibilities and attributes for each component. He then tried constructing a path for a simple robotic task (stacking blocks) and realized that his *Internal Power* component was no longer needed, since he had added *Motor-Power Arm* and *Motor-Power End-Effector* components. The unneeded component was deleted and the new path completed (Figure 11).

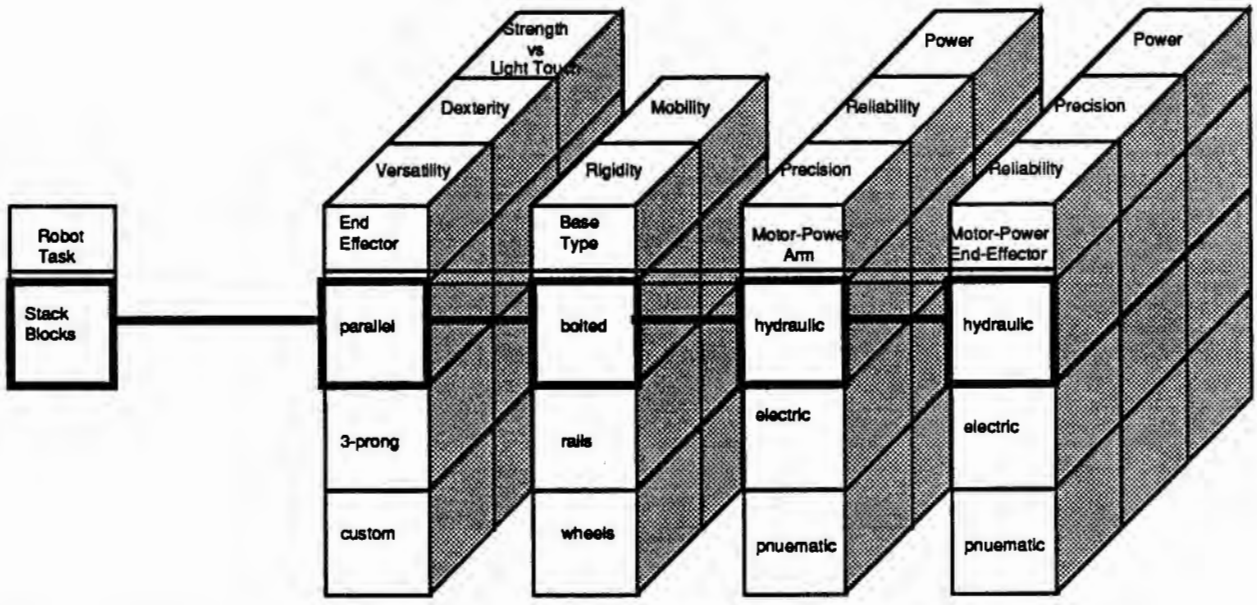


Figure 11. Selecting a Path for a Simple Task

As he was selecting a possibility from each component, he realized that some important components were missing. *Position Sensor*, *Arm Type*, and *Degrees of Freedom* were added, along with a set of possibilities and attributes for each component. The *Degrees of Freedom* component caused some discussion. Unlike the other components, this was not an actual part. However, the expert believed that it was an integral part of the design and should be included (Figure 12).

When asked to select a path for the design of a robot arm for use as a space station arm, the expert had little difficulty in choosing possibilities, with the exception of the *End Effector* component. None of the possibilities for that component were suitable, so he supplied a new possibility (Figure 13).

In the last exercise, the expert was given a partial path of possibilities from three components and asked if he could come up with a robot task naming such a path. Constructing a path from possibilities that are rarely used is a technique that can be used to ensure tht the design space is more fully covered (Figure 14).

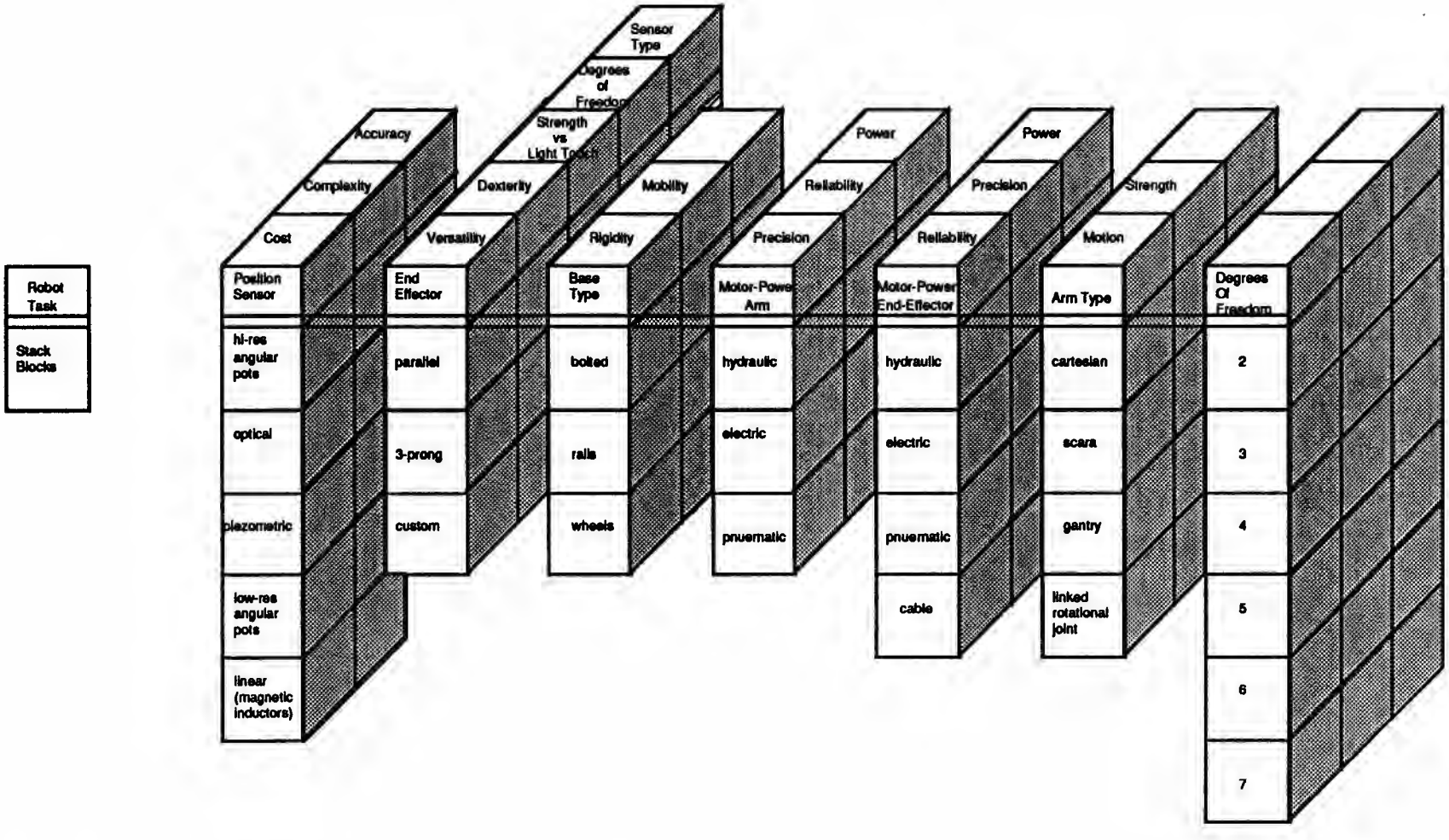


Figure 12. The possibility table after adding the components *Position Sensor*, *Arm Type* and *Degrees of Freedom*.

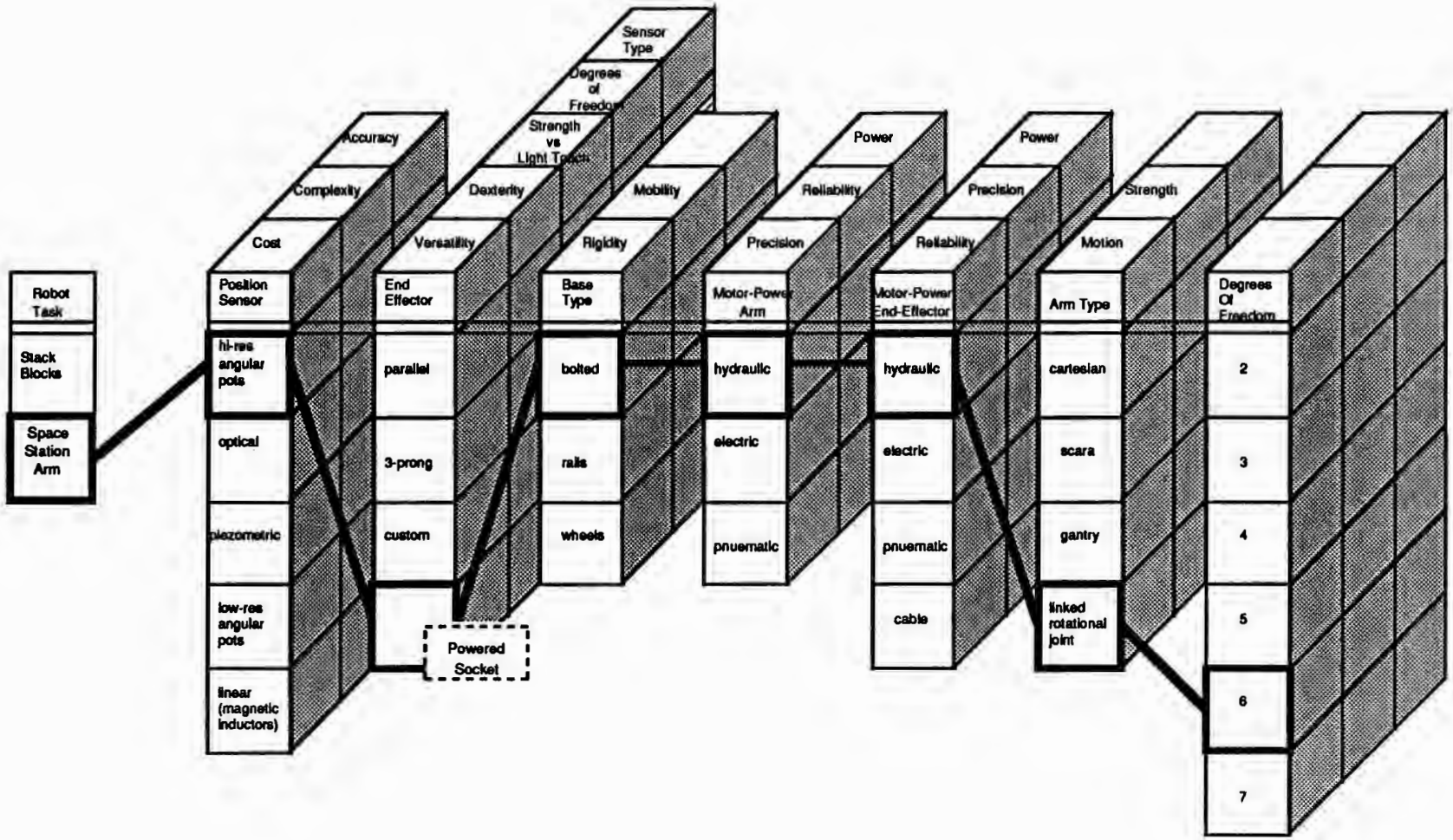


Figure 13. The engineer constructed this possibility path for *Space Station Arm.*

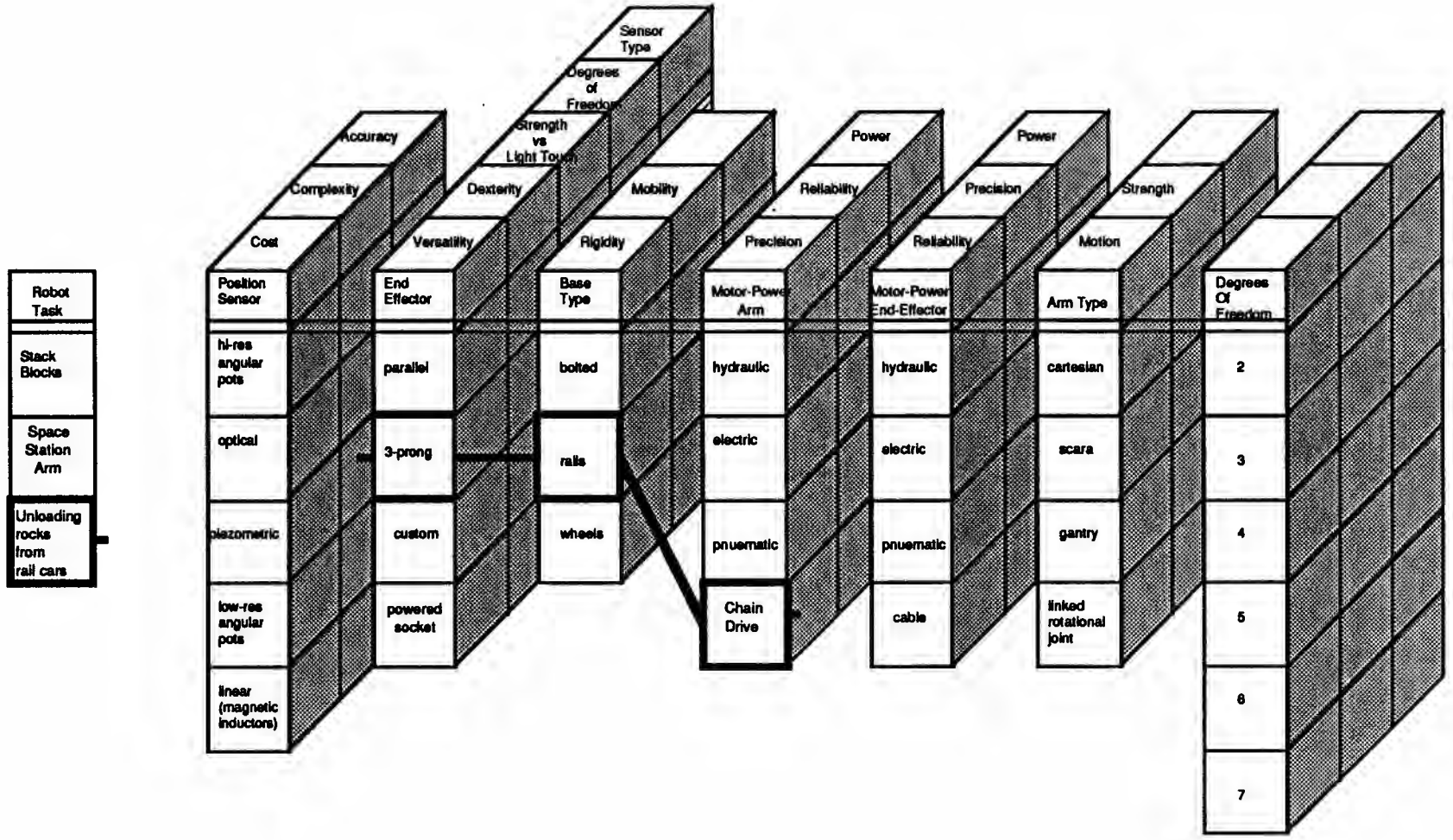


Figure 14. The engineer generates a new alternative consistent with a partial path proposed by the system.

2.3. APPLICATION OF *CANARD* TO NASA CORPORATE MEMORY FACILITY PROJECT

Under a NASA contract, the Boeing Advanced Technology Center is conducting research leading to a corporate memory facility (Boeing Computer Services 1989a,b,c). The goal of this project is to study and demonstrate techniques that could be employed to capture and use decision history and rationale throughout the life cycle of a major NASA program, such as Space Station Freedom. Within the Space Station Freedom program, we are examining aspects of the Power subsystem and the Environmental Control and Life Support subsystem. We are studying two areas to see if our ideas will apply across a breadth of applications, in different design stages, solving different types of problems.

We are focusing our Phase 1 research on design knowledge capture. NASA believes a Design Knowledge Capture tool should assist the engineer in providing a comprehensive definition of design knowledge including the rationale for the design chosen, specification of the design's performance, and functional breakdowns of the design showing dependencies and inter-relationships (NASA, 1988). As an important part of this project, *Canard* is intended to demonstrate the feasibility of using an automated tool to generate and capture some of the information being considered at various times during the design process.

3. STATUS AND FUTURE WORK

The current version of *Canard* contains the basic graphical components for constructing and displaying possibility tables and paths, a rudimentary link to *Aquinas* and *Axotl,* and a limited set of constraint tools. Future work will include enhancing these basic capabilities to the extent described in this paper and performing a thorough evaluation of the tools in the context of NASA design problems. We also contemplate the eventual implementation of other features, including:

Hierarchies of possibility tables. The alternatives and path attributes constructed at one level become components in a possibility table at a higher level of design.

Voice and graphic explanations. Additional information about the possibilities, alternatives, traits, and design goals can be added by the designer in text, graphics, or voice format.

Design change history. As changes are made to the paths during the process of making tradeoffs, *Canard* will record the changes and prompt the designer for reasons for the changes. Voice or text formats will be available.

Links to external databases or knowledge bases, and model-based reasoning tools. External databases containing component information and knowledge bases previously developed can be plugged into the possibility table alongside the *Aquinas* knowledge bases.

Links to Axotl influence diagram facility. Influence diagrams could be plugged into the back of possibility tables in the same manner as repertory grids, to assist in problems with significant risks or uncertainties or complex tradeoffs.

Multiple expert tools. Aquinas tools that assist multiple experts in reaching consensus could be integrated with the *Canard* facility (Boose and Bradshaw, 1987).

ACKNOWLEDGEMENTS

We express our appreciation to Miroslav Benda, Kathleen Bradshaw, Beverley Clark, Karolyn Covington, Mike Freeman, Peter Friedland, Randy Johnson, Cathy Kitto, Sandra Marcus, Guy Olney, Kish Sharma, Suzanne Shema, Greg Swietek, Lisle Tinglof-Boose, and Bruce Wilson for their contributions and support. *Canard, Aquinas, Axotl,* and *DDUCKS* are under development at the Advanced Technology Center, Boeing Computer Services in Seattle, Washington.

REFERENCES

BOEING COMPUTER SERVICES (1989a) Information to be Retained in a Corporate Memory (CMF TR1), NASA contract NAS2-12108.

BOEING COMPUTER SERVICES (1989b) Knowledge Representation for a Corporate Memory Facility (CMF TR2), NASA contract NAS2-12108.

BOEING COMPUTER SERVICES (1989c) Knowledge Representation and Trade Studies (CMF DEMO1), NASA contract NAS2-12108.

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

BOOSE, J.H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man-Machine Studies,* 23, 495-525.

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

BOOSE, J.H. (1988). A survey of knowledge acquisition techniques and tools. *Proceedings of the Third AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop,* Banff, Canada, November 1988. To appear in *International Journal of Man-Machine Studies.*

BOOSE, J.H. & BRADSHAW, J.M. (1987). Expertise transfer and complex problems: Using *Aquinas* as a knowledge-acquisition workbench for knowledge-based systems. *International Journal of Man-Machine Studies,* 26, 3-28.

BOOSE, J.H., SHEMA, D.S. & BRADSHAW, J.M. (1988). 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.

BRADSHAW, J. M. & BOOSE, J.H. (1989). Decision analysis techniques for knowledge acquisition: Combining information and preference models using *Aquinas. International Journal of Man-Machine Studies,* in press.

BRADSHAW, J.M., BOOSE, J.H., COVINGTON, S.P. & RUSSO, P.J. (1989). How to do with grids what people say you can't: The application of decision analysis methods in *Axotl* and personal construct methods in *Aquinas* to design problems. *Knowledge Acquisition: An International Journal,* in press.

BRADSHAW, J.M., COVINGTON, S.P., RUSSO, P.J. & BOOSE, J.H. (1989). Knowledge acquisition techniques for intelligent decision systems: Integrating *Axotl* and *Aquinas* in *DDUCKS. Proceedings of the AAAI Uncertainty Workshop,* August 18-20, Windsor, Ontario, Canada.

CLANCEY, W. J. (1984). Classification problem solving. *Proceedings of the National Conference on Artificial Intelligence,* Austin, Texas.

HOWARD, R. A. & MATHESON. J. E. (1980). Influence Diagrams. Reprinted in Howard & Matheson (eds.) (1984). *Readings on the Principles and Applications of Decision Analysis*. Menlo Park, California: Strategic Decisions Group.

KAHNEMAN, D., SLOVIC, P. & TVERSKY, A. (1982). *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press.

KELLY, G. A. (1955). *The Psychology of Personal Constructs*. 2 volumes. New York: Norton.

MARCUS, S. (1987). Taking backtracking with a grain of SALT. *International Journal of Man-Machine Studies*, 26, 383-398.

MCNAMEE, P. & CELONA, J. (1987). *Decision Analysis for the Professional with Supertree*. Redwood City, California: The Scientific Press, 1987.

NASA (1988). Process Requirements Document for Design Knowledge Capture, Space Station Program, March 28, NASA Parkridge Space Center, Reston, VA.

RUBINSTEIN, M. F. (1975). *Patterns of Problem Solving*. Englewood Cliffs, N.J.: Prentice-Hall.

WISE, J. A. (1985). Decisions in design: Analyzing and aiding the art of synthesis. In G. Wright (ed.) *Behavioral Decision Making: Theory and Analysis*. New York: Plenum Press.

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

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A Generic Knowledge Acquisition Environment for Conceptual and Ontological Analysis

Douglas Skuce
Wang Shenkang
Yves Beauvillé

Department of Computer Science
University of Ottawa
Ottawa, Canada, K1N 6N5

ABSTRACT

We describe a general purpose knowledge acquisition environment written in Smalltalk that runs on a SUN or Macintosh. In this paper, we describe its features in general and its use in particular in conceptual and terminological analysis either for systems under design, such as expert systems or other knowledge-intensive systems, or existing systems. The system, termed CODE, could be described as a "knowledge engineer's assistant". A highly graphic interface permits experimentation with definitions of *concepts*, arranged in an inheritance network in which many useful inheritance mechanisms are available (including some new ones). Several associated semantic subsystems, such as a first order logic system and a simple natural language system, permit various types of syntactic and semantic checks to be performed. The primary applications of the system are in general knowledge engineering, and in the closely related area of software engineering. A large scale application in software engineering is in progress.

1. INTRODUCTION

The literature on knowledge acquisition and elicitation is rapidly growing, indicating the importance of knowledge processing in general; we take the papers presented at the Banff conferences to be representative [Boose&Gaines88]. We note certain emerging trends and subareas, and will attempt to relate our approach to others in the literature which share some essential aspects of our approach and concerns. The particular aspects we are concerned with are:

1 generality of potential applications (for some system)
2 emphasis on clarifying conceptual and ontological structure (of some domain)
3 emphasis on terminological and linguistic control (in that domain)
4 necessity for a strong, yet general knowledge representation
5 necessity to assist the user with difficult conceptual analysis problems

On these points: (1) We have preferred to develop a general purpose system, which is very useful as is but which can be easily specialized for some particular application if necessary. (2,3) We believe that an important and often overlooked aspect of knowledge acquisition involves the careful design of the ontology and the equally careful choice of terminology. We claim: "*terminological confusion breeds conceptual confusion, and vice versa*" (4) Many knowledge acquisition systems rely on one or two relatively simple paradigms for capturing knowledge. However, while often very useful for a particular aspect of

knowledge, the paradigm may be completely unable to handle many other types of knowledge that are absolutely essential. Hence our system has a strong, flexible knowledge representation, which includes a full first order logic deduction system. (5) Finally, given the difficulty of conceptual analysis and the need to understand the representational issues and common errors and problems in conceptualization, our approach is to provide the knowledge engineer with feedback that catches as many conceptual, logical, and terminological errors as possible, and guides him toward correcting them.

We believe that the basis of most knowledge acquisition problems lies in the difficulty many people have with expressing their knowledge and thoughts precisely in natural language. But at this point in history we have little choice: usually knowledge is initially expressed or conceived of in natural language as ideas, informal discussions, or written text. Often this knowledge, if it is reexpressed in some more formal language like those that computers accept, is partially lost or distorted in the translation process. A closely related problem is that experts may in fact not have an adequately clear conceptual picture in their minds; the confusion is brought to light during the difficult process of knowledge acquisition. The methodology we have developed, termed *Conceptually Oriented Design/Description* (COD), is intended to assist with some of these linguistic and conceptual analysis problems. COD integrates knowledge representation ideas from artificial intelligence (ai), ideas from object oriented programming (oop), plus a modest component of natural language syntax and semantics. The ideas are derived from many years experience in knowledge representation and acquisition [Skuce77,83], including commercial work with major expert system tools like KEE and ART. In this paper, we first outline briefly the underlying ideas of COD, but then focus on the functionality of the knowledge processing system itself, termed CODE (COD Environment), in the context of knowledge acquisition. COD has its formal basis in abstract data types; this is discussed in [Skuce89c]. Recent related work is [Skuce89a,b,c]

In the usual context of knowledge acquisition, CODE may be thought of as a "knowledge engineer's assistant", or, as we prefer, since it emphasizes its use by other than knowledge engineers, "a spreadsheet for ideas". Using CODE, one can experiment with relationships between concepts, do "what...if" experiments, and obtain quick feedback on the desirability of changes and additions. The design of CODE reflects many conceptual, linguistic, and logical features that a wide variety of types of knowledge acquisition situations have in common. The features we have designed into CODE, which we describe in this paper, constitute a general purpose knowledge acquisition, debugging, and accessing system - a *knowledge processing system* - which could, for any particular application, be specialized further should its generic properties not be sufficient. The current design of the system does not reflect any particular application bias. We are currently beginning to apply it to areas such as software engineering and lexicon construction[1]. Our goal is to design CODE so that relatively little will need to be added for each application, i.e. we hope that the generic CODE system will indeed satisfy a broad collection of knowledge acquisition and representation requirements.

The paper is organized as follows. In Section 2, we briefly introduce the basic ideas of COD. This topic itself properly requires considerable space, and is discussed more fully in [Skuce89a]. The actual software components of the CODE system are described in Section 3. Section 4 explains further some of the examples and figures that have been used earlier. Section 5 discusses a number of other systems that have some aspects in common with

[1] These two domains seem, at first glance, unrelated, which illustrates an important point we cannot overemphasize: they are both deeply dependent on precise conceptual and terminological analysis, and hence can equally benefit from a tool like CODE.

Unlike most other inheritance systems, whose notion of property and inheritance is more restricted than ours, any expression that is meaningful to either humans or any associated CODE subsystem can be a property: the interface and inheritance part of the system treats properties just as textual expressions. How they are subsequently processed, both syntactically and semantically, depends on which category they are from and what processing capability exists for this category. (The two subsystems currently available for semantic processing, for ClearTalk and for logical verification, are described in Section 3 below.) The modes of inheritance are controlled by *flags*, which determine how it behaves in inheritance. We distinguish two senses of "a property is inherited:" 1) a property is *inherited by name* if subconcepts have such a property with the same meaning but whose body bears no relation to the "higher" body; 2) it is *inherited by body* if the body is a logical derivative of the higher body, or at least defaults to the higher body. The following flags control the most basic inheritance behaviour, regardless of other flags which may be compatible.

Basic flags (mutually exclusive)

private — the property does not inherit at all
Ex: any definition

class — the property inherits to subconcepts, but not to instances
Ex: the number of instances of a class (inherited by name to all subclasses)

instance — the property inherits both to subconcepts and to instances
Ex: the age of a person

Body flags (control what happens to the *body* during inheritance):

has — the property inherits only by name, i.e. with a nil body, which must be redefined in each subconcept. Hence 'h' means only that such a property exists.
Ex: name of a person

modifiable — the property inherits by body, but the body of the property may be changed in any way in subconcepts (the inherited value is the default)
Ex: the 'flies' property of a bird (the default; some birds don't fly)

refinable — modifiable in subconcepts, but only in a logically consistent manner
Ex: the age of a person is an integer between 0 and 115 (a subconcept like 'child' must be consistent with this)

fixed — the property inherits by body, but the body *cannot* be changed in subconcepts
Ex: the constraint 'is alive' in the concept 'living things'

Logical flags (specify additional *logical* behaviour):

necessary — the property is necessary
Ex: the father of a person (i.e. everyone has a father)

sufficient	the (constraint) property is sufficient for determining membership in the class, given that it is in the superclass Ex: the constraint 'age > 17' for the subclass 'adult' of 'person'
optional	the property is optional Ex: the attribute: 'job' for a person
typical	the property is typical (implies optional) Ex: the constraint 'can fly' for a bird, i.e. a typical bird can fly

We believe that this notion of "flag" is quite unique to our system. The following are some examples of complete property expressions (flags are abbreviated by one letter):

```
age (rin):                  an integer          ; the age attribute of a person
sex(rin):                   oneOf male, female  ; the sex attribute of a person
age limit (rin):            0 < age < 120       ; an age constraint on a person
rule 1 (fin):               married iff ~single
                                                ; a rigid constraint for married persons
flies (mit):                yes                 ; one way of saying that birds typically fly
                                                  could be 'no' in subconcepts
def(p):                     child and male      ; definition of 'boy'; subclasses and instances
                                                  do not have definitions
no of instances (hc):  an integer               ; the number of instances of a concept
cd name(hin):                                   ; all subconcepts must have some name
comment (hio):                                  ; all subconcepts will inherit an empty
                                                  comment, which may be filled in
```

The flags may appear in any order in parentheses and the sources are not shown here, but they may be on the screen. The user may add new flags for whatever special purposes, and define additional inheritance rules. The system checks to maintain certain logical relations between the flags correctly. The aspects of the system we have described above are the main ones. There are several other modes of inheritance, such as "blocking" inheritance, and "compound properties", and other kinds of flags and properties that we have not described for lack of space.

3. SYSTEM COMPONENTS

Figure 1 gives an overall block diagram of the system, which is written in Smalltalk 80 and Prolog. The user interface consists of three main components: the cd views, the grapher, and the property browser. Figure 2 shows an overall image of the complete SUN screen on which these can be seen. The various subsystems used for syntactic and semantic checking are invocable from the interface.

3.1. User Interface Components

3.1.1. Cd Views

The primary interface components are the *cd views* which show cds; a typical example is shown in Figure 3. The user creates a cd by a mouse menu item. The visual appearance of the cd view is controlled by a number of *view* properties which the user may define and which inherit normally. Any aspect of any cd view may be updated at any time. The top

latter by entering these concept names under 'kinds:" in the system property subview of the cd for 'student'. Should we wish to logically check any properties of students, the following wffs can be created automatically and sent to the logical subsystem:

person(X) if student(X)
undergraduateStudent(X) iff student(X) and ~graduateStudent(X)
graduateStudent(X) iff student(X) and ~undergraduateStudent(X)
undergraduateStudent(X) or graduateStudent(X) iff student(X)

This expresses the fact that students are persons and that the two subconcepts of student are in fact a partition, i.e. disjoint and exhaustive. Knowing this, the deduction system can catch any redundancies or outright contradictions we might make later. Any *other* wffs that the user has explicitly created may also be passed to the logical system.

The logical subsystem is written in Prolog (to attempt to program it in Smalltalk would be difficult and inefficient), and communicates to the interface via Unix sockets (on the SUN), which permits it to run concurrently on a different machine. The user may open a view in the interface that monitors the expressions being collected for submission to the logic system. When desired, a group of these may be processed by the logic system, and the user may then interact with it directly. In the logic system, one may check for *redundancy* or *contradiction* and *trace deduction*, either forward or backward.

The logical subsystem requires the user to have a more advanced knowledge of logic than the typical expert system or Prolog user may have, but we believe that this is unavoidable if the user is to be able to express or at least properly comprehend the logical meaning of such rules. Anyone who cannot does not have to use it. Instead, he could write rules in either English or ClearTalk, and have a logical expert translate and explain them. We have found however that even a logical expert can easily overlook possible contradictions. As an illustration, the reader is invited to write a set of definitions for marriage laws and to attempt to verify manually that they do not contain any contradictions or omissions.

4. EXAMPLE: PROCESSING KNOWLEDGE ABOUT FAMILIES

In this section, we discuss some aspects of commonsense knowledge about families acquired using the system and illustrated in the accompanying figures. Figure 2 shows the full SUN screen (2 million pixels) with a typical "scene" created by the user: the grapher view in lower left, the property browser beside it, and two cd views filling the remaining space. Super concept links are labeled 's' on the graph, and "kinds" groups are labeled 'k1', 'k2', etc. Links labeled with property names are optional on the graph; one may display only those links he is currently interested in, like 'has part'. There is one instance ('i'), 'Wang', who is a husband and a father. The browser has been set to show all properties of a person.

Figure 3 shows a cd in the "standard" format. A second "ClearTalk" format can be generated from the former; it is very readable English, suitable for verification or teaching purposes (see Figure 1). We can see that fathers have five attributes, five constraints, and three events. The symbol '#' is read 'self', in this case 'a father'. The document property 'source' is fictitious, and the 'purpose' is not given[1] (what is the purpose of a father?).

[1] The 'source' property means the document from which the knowledge was obtained, while the 'purpose' property is used as an indexing mechanism: it may be filled with ClearTalk expressions and searched on to find concepts with a certain purpose.

Figure 4 shows several uses of the property browser. The top bar is clicked on in either half to select the mode, and subsequent choices are made left to right. Each subview has a pop up menu, e.g. one may add, delete, rename, move, etc., after selecting a particular property. The lower right subview reproduces the system property subview from the cd currently selected.

Figure 5 shows a typical interaction with the logical subsystem, programmed in Prolog. A number of rules, both implicit and explicit, have been collected from various cds and sent to the subsystem for analysis. (We are in the process of automating this operation.) Here we see a number of rules relating to allowed marriage and the associated definitions of family relationships. This example seems trivial, but we were nevertheless surprised by some of the errors we found in our initial set of rules, too long to include here. The bottom window shows the response of the system when it finds that Ann and Mary cannot marry because they are both female.

5. RELATED SYSTEMS

The six knowledge acquisition workshops in both North America and Europe, devoted entirely to this subject, have attracted much interest [e.g. Boose&Gaines88]. J. Boose, the co-organizer (along with B. Gaines) of these workshops, has written a survey of the field in the most recent of these [Boose88]. A very comprehensive list of systems, with brief descriptions and an extensive bibliography, is also provided in the article. The articles by Kitto [Kitto88] and Gaines [Gaines87] are also particularly valuable. Boose partitions knowledge acquisition systems in several ways, but we would suggest that knowledge acquisition systems may also be partitioned in the following two ways. The first partition is by *purpose*, into those intended specifically for typical expert system development applications vs. those intended for other specific or even general purpose use. The second partition is by *mode of use* : those we shall term *elicitation* systems, where the computer-naive expert sits, possibly alone, in front of the system and responds to questions (probably similar to Boose's *interview* systems), vs. those we shall term *knowledge engineer's assistant (kea)* systems, intended for the experienced knowledge engineer (ke). The CODE system is intended for general, kea use, though the interface is very usable by persons who are not kes. In our preferred use of the term, elicitation systems concentrate on attempting to elicit knowledge from an expert who does not understand any principles of knowledge representation. (We assume that, for all but trivial problems, a ke must participate at some stage of knowledge acquisition and refinement for whatever purpose.) We first review some systems with similar features to CODE, and then discuss some of the trends noted in the introduction.

The following kea systems are typical. The KREME system [Abrett&Burstein88], developed at Bolt Beranek and Newman, runs on a Lisp machine and is intended strictly for expert ke use. It does not appear to have any natural language or logical processing ability, and its notion of property is that of a conventional "frame" system, i.e. essentially what we have termed attributes. It permits graphic display of inheritance nets, and rapid editing of properties, although its forms of properties and modes of inheritance are much more restricted than in CODE. It is intended to be part of a general knowledge representation system to be used in various ai systems.

The ONTOS system, developed at Carnegie Mellon University, [NirenburgEtAl88] is also Lisp based. It is quite similar functionally to the KREME system, differing mainly in that it uses a very flexible (but quite low level) frame representation language that permits considerable flexibility in inheritance schemes. It has been used to acquire a considerable

body of knowledge (thousands of concepts) on various subjects such as the IBM XT, and again is intended for kea, general purpose use, particularly in conjunction with their extensive natural language processing projects.

The KEATS system, developed at The Open University [MottaEtAl88], runs on a Lisp machine. It is designed to assist in processing natural language transcripts of interviews with experts to aid the ke in extracting knowledge. The expert's knowledge is assumed to be captured, at least in part, in the interview transcripts. KEATS integrates two very useful aids: tools for segmenting and analysing the transcript in a "hypertext" mode, and tools for drawing semantic networks of the relations between frames in the knowledge base. The underlying knowledge representation is a conventional rule and frame-based approach.

The CYC system, developed at MCC [LenatEtAl86], [Lenat&Guha88], has attracted considerable attention due to its ambitious scope. It is a general purpose kea system, intended to capture a very wide assortment of general "commonsense" knowledge about the real world. Running on Lisp machines, it particularly addresses the problems of assembling a very large knowledge base (potentially millions of knowledge units) whose evolution must extend over many years and many kes.

The SB-ONE workbench [Kobsa89] is a large, comprehensive system which is integrated with an expert system and with a natural language front end. Its semantic structure is based on the KL-ONE knowledge representation family [Brachman&Schmolze85]. Also Lisp-machine based, it has been applied to developing a system for explaining insurance forms to applicants.

We next point out some research reported at the 1988 Banff meeting that reflects some of the concerns for certain aspects of knowledge acquisition we noted in the introduction. First, there is increasing concern to include some ability to deal with natural language. Some systems assist in analysing unrestricted text [MottaEtAl88, Linster88; Breuker&Wielinga89, Slator88], although to do this automatically involves all the difficulties of natural language understanding [NirenburgEtAl88, Szpakowicz88]. Many of these systems however reduce natural language to typical knowledge representation structures which may *lose* the original language; our goal is to *retain* just the essential amount, and ensure that it is linguistically sound.

Concern for a generalized approach has been demonstrated by [NirenburgEtAl88] and [Murray&Porter88], to name two. A closely related concern is that of acquiring what could be termed a reasonably complete *model* (in general, a conceptual model, or in the case of a specific physical system, more like a simulation model) of the subject, i.e. a more complete representation as opposed to just a collection of rules. Doing this leads eventually to serious ontological analysis, but few systems have done much in this area (CYC and ONTOS are notable exceptions). [Musen88], [Slator88] and [Regoczei&Hirst88] all point to the importance of conceptual modeling or ontological analysis.

4. DISCUSSION AND FUTURE PLANS

4.1. CODE as a Highly Generic Knowledge Processing Systeem

CODE has the following quite unique features which distinguish it from many of the other systems reviewed above:

- it is intended for a very wide range of applications
- it can be used by computer novices, but not optimally
- it is highly portable with modest machine requirements
- it has the associated ClearTalk subsystem for basic natural language analysis
- it has the associated logic subsystem for full first order logical analysis
- it has a very flexible notion of property and inheritance, and is not committed to some particular interpretation of properties

Clearly there are many approaches to implementing a knowledge processing system. Many of the systems that have been developed are closely coupled to expert system applications, which are usually oriented toward capturing the knowledge needed for some quite specific application, such as diagnosis. Much emphasis has been placed on acquiring *strategic* or *task-solving* knowledge, usually in the form of rules, and sometimes at the expense of other types of knowledge such as ontological, terminological and logical (i.e. full first order, as opposed to the typical simple "logic" of many expert systems).

CODE is intended for a wide range of general purpose applications in capturing a large fraction of knowledge usually expressed in natural language specifications, descriptions, or definitions. Its design philosophy has kept it as generic as possible, on the assumption that 1) it can be used as is in applications for which there is currently available no other more specific knowledge processing tool, or 2) it can be specialized relatively easily for specific applications like typical expert systems or software engineering. The specialized versions of CODE will of course inherit its unique properties that many of today's specialized tools lack, such as the emphasis on language and full logic.

4.2. Applications of CODE

At the time of writing (August 1989), the system has been used to describe part of UNIX (a perfect application for COD), a well known database design problem, and is now being used to specify CODE itself. We anticipate that a reasonably complete description of CODE will involve about 100 to 200 entities, and perhaps 50 to 100 situations (event or state verbs). Our purpose in using it to document itself is threefold: to illustrate its utility for software documentation (both user and system maintenance), to aid its users (an on-line "intelligent help"), and to assist in developing future versions.

A second application, illustrating the flexibility of CODE, is in constructing a knowledge base for use by terminologists (persons who compile lists of terminology in some domain, rules for its use, typically related to translation between languages). A base of several hundred concepts in the field of intelligent tutoring systems is being constructed by two terminologists, who have no "ai" training.

A third application is in software engineering in an industrial setting. CODE is being used to capture a comprehensive description of a large case tool in final stages of development. (It is unfortunate that it could not have been involved from the outset.) This knowledge base will serve as a basis for a) documentation, b) maintenence, c) future evolution, and d) training activities on this system. It will replace or control hundreds of pages of otherwise poorly structured and terminologically confused natural language documentation.

We believe that knowledge engineering and software engineering have much more in common than is often acknowledged. It is our intention in the near future to focus in particular on applications of COD in software engineering. We intend to develop COD for use throughout the software life cycle, from requirements analysis to end-user support. At the *design* end, the CODE system permits the intentions of the customers and/or designers

to be captured and debugged at the conceptual level, before any code or even any commitment to a programming language and environment is made. The conceptual structure and the terminology of the proposed system can be laid out clearly, like initial drawings for a physical system such as a building. Logical and terminological confusion can be eliminated before any code is written. Subsequent programming can be driven directly from the cds themselves, most naturally using an object oriented programming system (see below). The *user* of a system designed using COD would benefit from COD in the form of intelligent on-line help, i.e. by being able to access a COD description of the system, i.e. to "see" the designer's intentions. We plan to couple CODE to the Eiffel system [Meyer88], a new object oriented compiler.

4.3. Future Enhancements

We plan the following enhancements in the immediate future.

1. The ClearTalk system will be more closely coupled to the user interface, so that every word or phrase can be immediately checked and entered in the lexicon. The lexicon will be enhanced so that every occurrence of an entry can be located. This will give the system a more "hypertext-like" behavior, seen already in some existing tools. The linguistic coverage of ClearTalk will also be extended, so that it approximates English better. Linguistic knowledge like morphology and synonyms will be available.

2. The logic subsystem will similarly be more closely coupled, so that it will be continually available in the background. A simple mouse click in a cd will incrementally compile and process new rules and facts, and as soon as any problem is detected, the logic system will pop up a window and request intervention. The system will then list a number of suggested ways of alleviating the problem. The user may choose one, or may have another in mind. This change will then be automatically reflected in the cd knowledge base.

3. The graphic system will be enhanced, so that graphs with hundreds of cds can be managed without excessive clutter. In the current system there is some ability to display only subsets of nodes or arcs, based on various criteria, but this can be improved.

4. The ability to access multiple knowledge bases simultaneously will be developed. Closely related to this is the problem of sharing multiple knowledge bases between workers, who are all in the process of modifying them. The experience of the CYC project, one of very few to address this problem, should be useful.

5. A widely applicable ontology will be developed. By this we mean an organization of very general concepts useful for most world knowledge classification at very high levels. At the moment, every general purpose system like CODE has to develop its own ontology. It is our belief that many of these are very similar, and that in time, a generally agreed upon set of high-level semantic categories (like *entity*, *situation*, *attribute*, etc., and their immediate subcategories) will be developed. This program is clearly contentious at the moment, however.

6. While CODE is not intended to be a learning system per se, we have begun to experiment with several simple ways in which it can suggest possible generalizations from the cds it knows about. For example, if the same property is put into two cds separately, the system asks if it should possibly be placed in their common parent, if there is one. We would also like to add some ability to measure "similarity" between concepts.

ACKNOWLEDGEMENTS

This research has been directly supported by Cognos, Inc., and the URIF program of the Ontario government. It is derived from earlier research supported by Cognos, The Natural Sciences and Engineering Research Council of Canada, and the National Research Council of Canada. Mr. Wang Shenkang has been supported in part by the Government of the People's Republic of China (Pao Zhaolong Scholarship Fund). Ingrid Meyer has provided many valuable suggestions both on the linguistic aspects of the system and on this paper.

REFERENCES

[Abrett&Burstein88] G. Abrett and M. Burstein, The KREME Knowledge Editing Environment, *Int. J. Man-Machine Studies*, vol. 27, pp. 103-126, 1988.

[Boose88] J. Boose, A Survey of Knowledge Acquisition Techniques and Tools, *Proc. 3rd Knowledge Acquisition Workshop*, Banff, Alberta, November 1988.

[Boose&Gaines88] J. Boose and B. Gaines, *Proc. 3rd Knowledge Acquisition Workshop*, Banff, Alberta, November 1988.

[Brachman&Schmolze85] R. Brachman and J. Schmolze, An Overview of the KL-ONE Knowledge Representation System, *Cog. Sci.*, vol. 9 no. 2, pp. 171-216, April 1985.

[Breuker&Wielinga88] Breuker, J. and Wielinga, B., Structured Knowledge Acquisition for Expert Systems, Proc. Expert Systems and Their Applications, Avignon, 1985.

[Gaines87] B. R. Gaines, An Overview of Knowledge-Acquisition and Transfer, *Int. J. Man-Machine Studies*, vol. 26, pp. 453-472, 1987.

[Kitto88] C. Kitto, Progress in Automated Knowledge Acquisition Tools: How Close are We to Replacing the Knowledge Engineer?, *Proc. 3rd Knowledge Acquisition Workshop*, Banff, Alberta, November 1988.

[Kobsa89] A. Kobsa, The SB-ONE Knowledge Representation Workbench, *Proc. of the Workshop on Formal Aspects of Semantic Nets*, J. Sowa (ed), Catalina, Feb., 1989.

[LenatEtAl86] D. Lenat, M. Prakash, and M. Sheperd, CYC: Using Commonsense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks, *The AI Magazine*, vol. 6, no. 4, pp. 65-85, 1986.

[Lenat&Guha88] D. Lenat and R. Guha, The World According to CYC, MCC Tech. Report ACA-AI-300-88, Sept. 1988.

[Linster88] Linster, M., A Critical Look at Kriton, Proc. 3rd Knowledge Acquisition Workshop, Banff, Alberta, Nov, 1988.

[Meyer88] B. Meyer, Eiffel: A Language and Environment for Software Engineering, *J. of Systems and Software*, vol. 8, pp. 199-246, 1988.

[MottaEtAl88] E. Motta, M. Eisenstadt, K. Pitman, and M. West, Support for Knowledge Acquisition in the Knowledge Engineer's Assistant (KEATS), *Expert Systems*, vol. 5, no. 1, February 1988.

[Murray&Porter88] Murray, K., and Porter, B., Developing a Tool for Knowledge Integration: Initial Results, Proc. 3rd Knowledge Acquisition Workshop, Banff, Alberta,Nov, 1988.

[Musen88] Musen, M., An Editor for the Conceptual Models of Interactive Knowledge Acquisition Tools, Proc. 3rd Knowledge Acquisition Workshop, Banff, Alberta, Nov, 1988.

[NirenburgEtAl88] S. Nirenburg, I. Monarch, T. Kaufmann, I. Nirenberg, and J. Carbonell, Acquisition of Very Large Knowledge Bases: Methodology, Tools, and Applications, Carnegie Mellon University Tech. Report CMU-CMT-88-108, 1988.

[Regoczei&Hirst88] Regoczei, S. and Hirst, G. , The Meaning Triangle as a Tool for the Acquisition of Abstract, Conceptual Knowledge, Proc. 3rd Knowledge Acquisition Workshop, Banff, Alberta, Nov, 1988.

[Skuce77] D. Skuce, Toward Communicating Qualitative Knowledge Between Scientists and Machines, PhD dissertation, Dept. of Electrical Engineering, McGill University, Montreal, 1977.

[Skuce83] D. Skuce, The LESK Tutorial, Dept. of Computer Science, University of Ottawa, Dept. of Computer Science Tech. Report, no. TR-83-03, 1983.

[Skuce89a] D. Skuce, A Language and System for Making Definitions of Technical Concepts, To appear in *Journal of Systems and Software*, 1989.

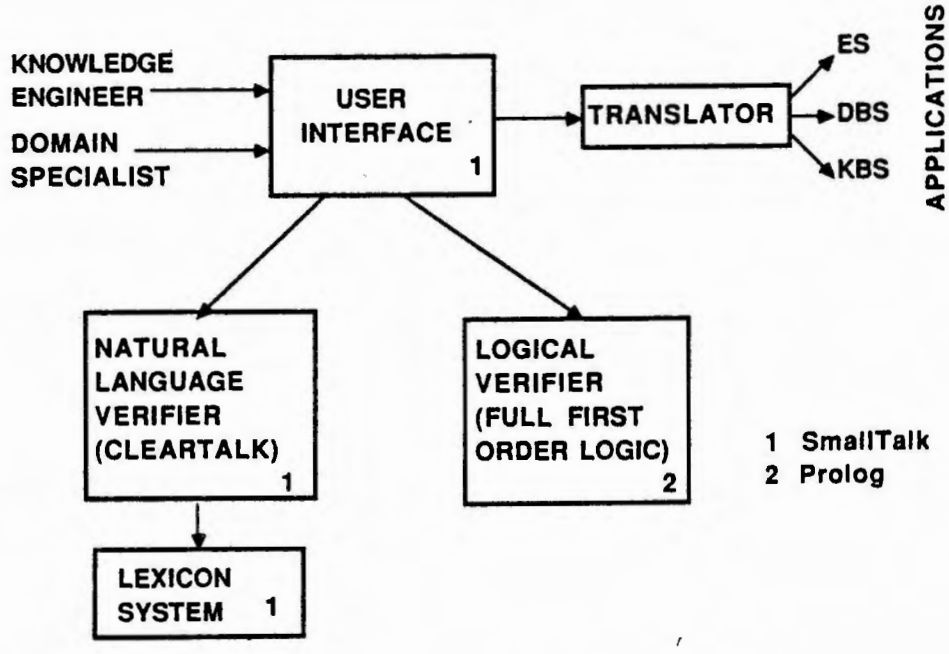
[Skuce89b] D. Skuce, A Knowledge Acquisition Language Based on Order Sorted Algebra, To appear in *Formal Aspects of Semantic Networks*, J. Sowa (ed), Morgan Kaufman, 1989.

[Skuce89c] D. Skuce. Beyond Objects: A Synthesis of Objects, Logic, and Natural Language for Software Production. Submitted to: *IEEE Transactions on Knowledge and Data Engineering*.

[SkuceEtAl85] D. Skuce, S. Matwin, B. Tauzovich, F. Oppacher, and S. Szpakowicz, A Logic-based Knowledge Source System for Natural Language Documents, *Data and Knowledge Engineering*, vol. 1, pp. 201-231, 1985.

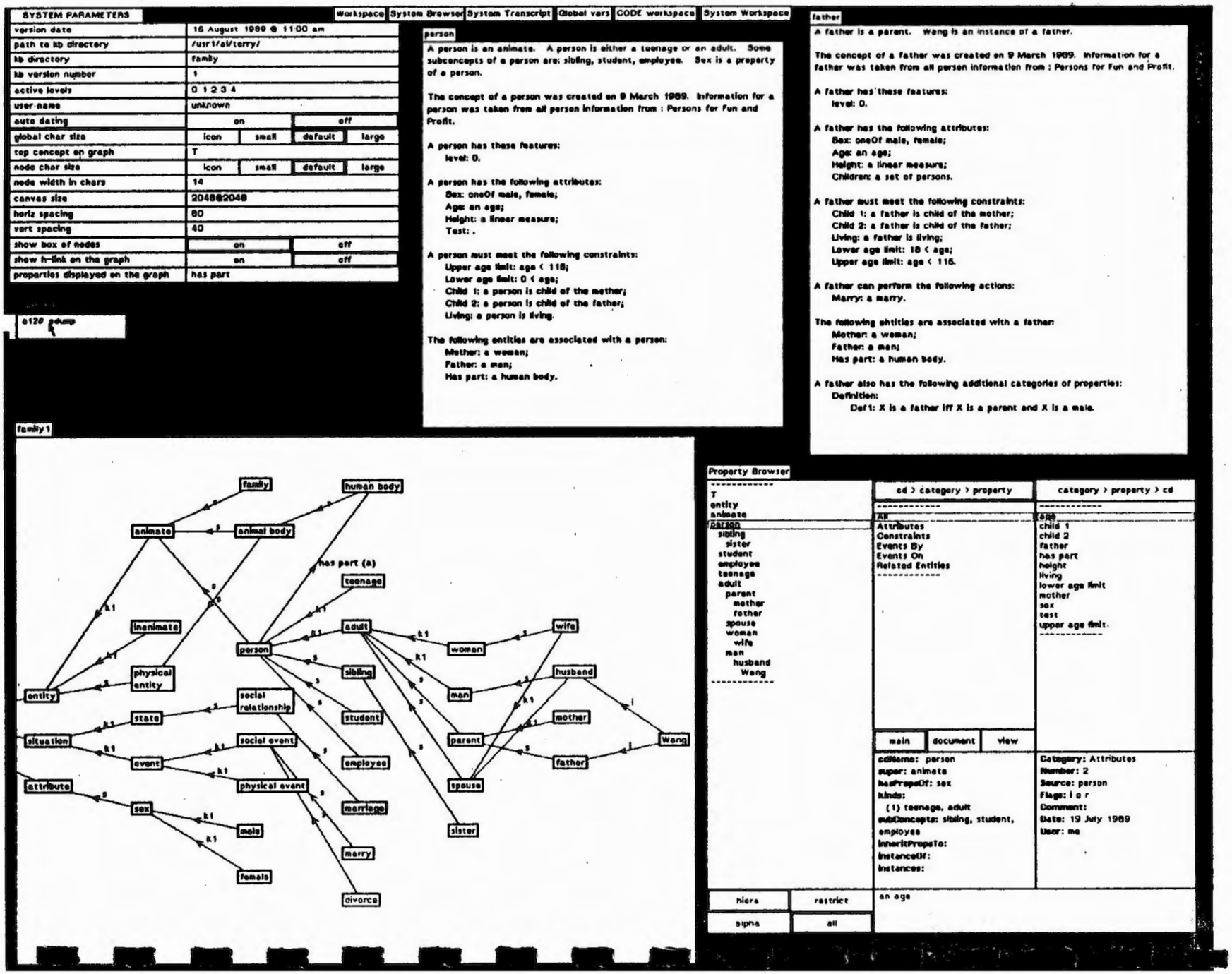
[Slator88] Slator, B.,Extracting Lexical Knowledge from Dictionary Text, Proc. 3rd Knowledge Acquisition Workshop, Banff, Alberta, Nov, 1988.

[Szpakowicz88] Szpakowicz, S.,Semi-automatic Acquisition of Conceptual Structure from Technical Texts, Proc. 3rd Knowledge Acquisition Workshop, Banff, Alberta, Nov, 1988.



1 System block diagram

2 Typical layout of SUN screen. One grapher view, one property browser view, and two cd views are open. The browser shows all the properties of 'person'



21-17

cdName: father super: parent hasPropsOf: kinds: subConcepts: inheritPropsTo: instanceOf: instances: Wang	General information date <T> : 9 March 1989 done by <T> : last updated <T> : comment <entity> : ; # is a concept that can exist, and is only denoted by a noun	source <person> : all person information from : Persons for Fun and Profit purpose <T> :	level <T> : 0

ATTRIBUTES
sex <sex> : oneOf male, female
age <person> : an age
height <person> : a linear measure
children <parent> : a set of persons

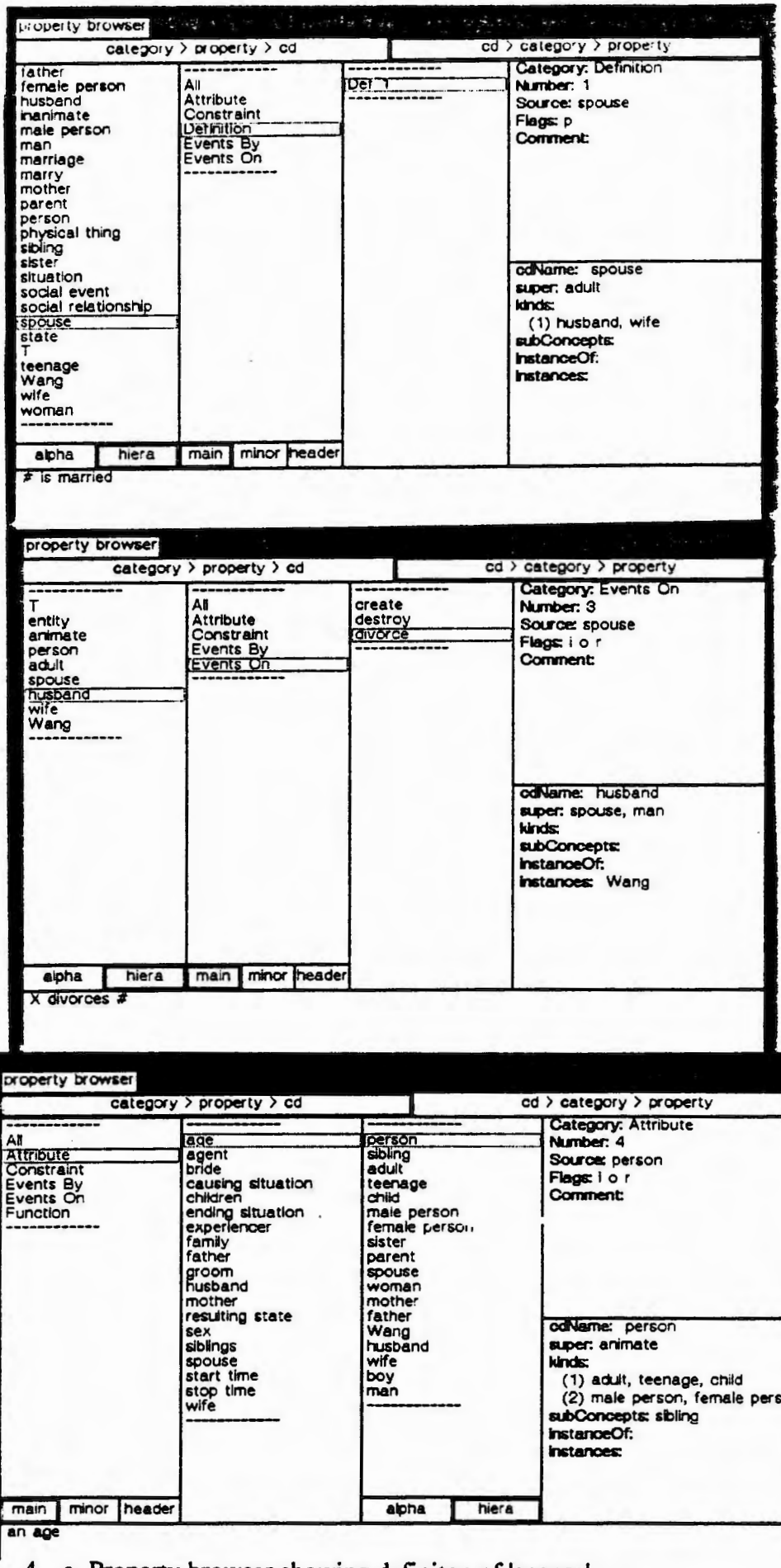
CONSTRAINTS
child 1 <person> : # is child of the mother
child 2 <person> : # is child of the father
living <animate> : # is living
lower age limit <adult> : 18 < age
upper age limit <person> : age < 116

EVENTS ON
marry <adult> : a marry

DEFINITION
def 1 <father> : X is a father iff X is a parent and X is a male

RELATED ENTITIES
mother <person> : a woman
father <person> : a man
has part <person> : a human body

3 A close up look at the cd for 'father'



4 a Property browser showing definiton of 'spouse'
b " " showing events on 'husband'
" " showing all attributes and body of 'age of person'

```
textedit - ex_smallFam, dir: /usr1/ai/yves/prolog/contradiction

rule(father(X,Y) iff parent(X,Y) and male(X)).
rule(mother(X,Y) iff parent(X,Y) and female(X)).

rule(parent(X,Y) iff child(Y,X)).

rule(son(X,Y) iff child(X,Y) and male(X)).
rule(daughter(X,Y) iff child(X,Y) and female(X)).

rule(brother(X,Y) iff son(X,Z) and child(Y,Z) and ~eq(X,Y)).
rule(sister(X,Y) iff daughter(X,Z) and child(Y,Z) and ~eq(X,Y)).

rule(~canMarry(X,Y) if sex(X,SX) and sex(Y,SY) and eq(SX,SY)).
rule(canMarry(X,Y) iff canMarry(Y,X)).

rule(sex(X,female) iff female(X)).
rule(sex(X,male) iff male(X)).
rule(female(X) iff ~male(X)).

fact(mother(ann,john)).
fact(father(bill,ann)).
fact(child(joe,bill)).
fact(sister(mary,joe)).
```

```
cmdtool - /bin/csh
Assert a fact (e for end) :canMarry(ann,mary).
Assert : canMarry(ann,mary) is a fact.

CONTRADICTION : Conflicting fact
Fact : canMarry(ann,mary)
contradicted by :
~canMarry(ann,mary) if sex(ann,female) and sex(mary,female) and eq(female,female)
  sex(ann,female) if female(ann)
    female(ann) if mother(ann,john)
      mother(ann,john) is a fact.
  sex(mary,female) if female(mary)
    female(mary) if daughter(mary,sk_4(mary,joe))
      daughter(mary,sk_4(mary,joe)) if sister(mary,joe)
        sister(mary,joe) is a fact.
  eq(female,female) is a fact.
```

5 Logical subsystem views: top, the current list of rules and facts; bottom, a contradiction detected resulting from the above rules and facts

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Using a Knowledge Acquisition Tool to Test Strategy Variations

Jeffrey Stout, Linda Green, and Sandra Marcus
Advanced Technology Center
Boeing Computer Services
P.O. Box 24346, MS 7L-64
Seattle, WA 98124

Abstract

This paper describes the use of a knowledge acquisition tool, SILICA, to create and support testbeds for experimenting with variations in problem-solving strategy. SILICA stores a declarative representation of domain facts that capture crucial relationships among attributes of domain objects, including the attributes of the solution. While most aspects of domain-knowledge use are fixed by the user at the time the knowledge is entered, some details are easily modified with potentially drastic effects on the generated system's performance. This paper will describe two kinds of strategy variations that can be set by the SILICA user. It will then report on empirical testing of the effects on the performance of a system that schedules equipment and personnel in a flight simulation laboratory.

1. Introduction

Last year's KA-for-constructive-problem-solving group identified the acquisition of strategic knowledge as one of the major difficulties in automating knowledge acquisition for tasks requiring constructive methods [Boose 89]. The difficulty is that the domain problem is often so complex and involves interrelationships among such a vast number of attributes that it is hard to know what strategy is appropriate without diving in and trying some strategy out. The cost of doing this "by hand" is so prohibitive that often a strategy is constructed according to what looks reasonable ahead of time with little experimentation on alternatives.

This paper describes the use of the knowledge acquisition tool SILICA to create and support a testbed for experimenting with variations in problem-solving strategy. SILICA stores a declarative representation of domain facts that capture crucial relationships among attributes of domain objects, including the attributes of the solution. Being facts, these aspects of the domain are fixed by the user at the time the knowledge is entered. SILICA also gathers information about the strategy the expert uses to reach a solution, and these details *can* be modified, with potentially drastic effects on the generated system's performance. This paper describes two kinds of strategy variations that can be set by the SILICA user. It will then report on empirical testing of the effects on the performance of a system that schedules equipment

and personnel in a flight simulation laboratory.[1]

Every week the Boeing Commercial Airplane Flight Simulation Laboratory (SIMLAB) receives approximately 200 requests to use configurations of simulation equipment for sessions in the following week. Requests come from task teams, usually identified with one programmer and one engineer, for blocks of specified lengths, for example three shifts of four hours each. Requests may include restrictions on the availability of members of the task team. Each request also specifies the kinds of equipment needed, including an airplane cabin; cabin peripherals such as visual displays, instrument panels, joysticks, armrests, and others; a mini-computer; and recording equipment such as stripchart recorders and tape recorders. The cabin peripherals and recording equipment can be switched from one configuration to another. The scheduler must assign equipment to each request within an appropriate time block. As well as attending to equipment availability and trying to maximize equipment use, the scheduler must be concerned with the schedules of individual task team members. Task teams prefer to have their sessions spaced out throughout the week so that they can analyze and react to data from each session before they run again. Individuals can also participate in more than one team so the scheduler tries to produce humane work schedules for individuals that allow, for example, for a reasonable amount of sleep each night (the lab operates seven days a week from 7 AM to midnight).

Scheduling the Flight Simulation Laboratory is difficult because it requires juggling a large number of soft constraints, all of which are important to some degree but which may be traded off against one another. A solution must achieve the right balance among these preferences as well as guarantee that all hard constraints are met. Our domain expert finds it easy to supply base knowledge that says what the constraints are and how they restrict the solution [Marcus 89]. How to juggle the constraints is harder to specify. The harder-to-specify strategic knowledge determines the order in which the constraints should be considered and how their impact should be taken into account.

Section 2 introduces the SILICA user interface to give an idea of how SILICA collects the base knowledge and how it allows some parameters of the strategy to be set by the user. Section 3 presents two kinds of variations in the way that base knowledge can be used. Section 4 gives our plan for testing the effectiveness of the strategy variations. Section 5 concludes with our statement of current status and expected status at the time of the 1989 Banff workshop.

2. SILICA's Collection of Knowledge

SILICA is an automated knowledge acquisition tool for constraint-satisfaction tasks such as designing an artifact or constructing a production schedule. SILICA 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. SILICA maintains a permanent, declarative store of the knowledge base which is updated during interviews with the domain expert, analyzed for completeness, consistency and

[1]The empirical testing is currently underway. The results will be reported at the workshop.

soundness and used as the input to the compiler.

Like many other successful knowledge acquisition tools [Davis 77, Boose 84, Kahn 85, Eshelman 86, Klinker 87, Marcus 88a], SILICA gets its power by making a strong assumption about the problem-solving strategy that will be used by any expert system it creates. SILICA assumes a generic propose-and-revise problem-solving strategy for creating a design. SILICA-generated systems incrementally build up a description of a design by proposing values for design parameters, one parameter at a time. At the same time, the system 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 goes back and changes specific decisions that will make the design fit the constraints. A truth maintenance system [Doyle 79] removes any part of the proposed design dependent on changed values, and the process of extending the proposed design resumes. Critical aspects of the strategy that can drastically affect efficiency have to do with the way initial values are proposed and the conditions under which backtracking is initiated.

2.1. The Base 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

These comprise the basic building blocks out of which the performance system is built. SILICA allows users to enter knowledge piecemeal starting at any point. The grain size of the pieces corresponds roughly to the SILICA 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. SILICA keeps track of how the pieces are fitting together and warns the user of places where pieces might be missing or creating inconsistencies.

In any SILICA 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
```

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, SILICA 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.

The examples shown below are based loosely on the SIMLAB knowledge base. An example of

a completed schema for a PROCEDURE for determining a value for TILE FOR SHIFT OF REQUEST is shown below with prompts on the left and knowledge supplied on the right. The user is instructed to supply a procedure that includes all of the necessary considerations that determine the value for the variable.

Knowledge Piece (1)

1	Name:	TILE FOR SHIFT OF REQUEST
2	Precondition:	NONE
3	Procedure:	ALLOCATION
4	Candidate set:	BLOCK
5	Hard constraint:	INCLUDE TIME FROM REQUEST
6	Hard constraint:	EXCLUDE TIME FROM REQUEST
7	Hard constraint:	EXACT HOURS FROM REQUEST
8	Hard constraint:	EXACT DAFCL FROM REQUEST
9	Hard constraint:	EXACT ATAP FROM REQUEST
10	Hard constraint:	EXACT LAB-429 FROM REQUEST
11	Hard constraint:	EXACT MONITOR FROM REQUEST
12	Hard constraint:	EXACT GLARE-SHIELD FROM REQUEST
13	Hard constraint:	EXACT SIDESTICK-MCFADDEN FROM REQUEST
14	Hard constraint:	EXACT SIDESTICK-SLIDER FROM REQUEST
15	Hard constraint:	EXACT SIDESTICK-NEW-BOEING FROM REQUEST
16	Hard constraint:	EXACT SIDESTICK-OLD-BOEING FROM REQUEST
17	Hard constraint:	EXACT SIDESTICK-SAAB FROM REQUEST
18	Hard constraint:	EXACT ADDS FROM REQUEST
19	Hard constraint:	EXACT SPERRY-CART FROM REQUEST
20	Hard constraint:	EXACT VIEWPOINTS FROM REQUEST-E
21	Hard constraint:	CHOICE-SET WSI FROM REQUEST
22	Hard constraint:	CHOICE-SET CAB FROM REQUEST
23	Hard constraint:	CHOICE-SET RT FROM REQUEST
24	Hard constraint:	CHOICE-SET RT FROM PRINTER-PLOTTER
25	Hard constraint:	CHOICE-SET RT FROM XYPLOT
26	Hard constraint:	CHOICE-SET RT FROM MONITOR
27	Hard constraint:	CHOICE-SET RT FROM DAFCL
28	Hard constraint:	CHOICE-SET RT FROM ATAP
29	Hard constraint:	CHOICE-SET RT FROM DUAL-DRIVES
30	Hard constraint:	CHOICE-SET RT FROM STRIP-CHART
31	Hard constraint:	CHOICE-SET RT FROM LAB-429
32	Hard constraint:	CHOICE-SET RT FROM SIMCON
33	Hard constraint:	CHOICE-SET RT FROM H800
34	Hard constraint:	CHOICE-SET RT FROM H1200
35	Hard constraint:	EXCLUDE TIME FROM BOOKED-ENGINEER
36	Hard constraint:	EXCLUDE TIME FROM BOOKED-PROGRAMMER
37	Hard constraint:	CHOICE-SET VISUAL-1 FROM TWO-REQUESTED
38	Hard constraint:	CHOICE-SET VISUAL-2 FROM TWO-REQUESTED
39	Hard constraint:	CHOICE-SET RT FROM CAB
40	Hard constraint:	
41	Flexible constrnt w/import:	INCLUDE TIME FROM DEFAULT-HOURS , 1000
42	Flexible constrnt w/import:	EXACT VIEWPOINTS FROM REQUEST-D , 1000
43	Flexible constrnt w/import:	MINIMUM START-TIME FROM LATE-EARLY , 900
44	Flexible constrnt w/import:	MAXIMUM END-TIME FROM LATE-EARLY , 900
45	Flexible constrnt w/import:	MINIMUM START-TIME FROM LONG-DAY , 850
46	Flexible constrnt w/import:	MAXIMUM END-TIME FROM LONG-DAY , 850
47	Flexible constrnt w/import:	MAXIMUM END-TIME FROM PILOT , 800
48	Flexible constrnt w/import:	EXCLUSION-SET DAY FROM PILOT-WEEKEND , 800
49	Flexible constrnt w/import:	EXCLUSION-SET DAY FROM WEEKEND , 700
50	Flexible constrnt w/import:	EXCLUSION-SET DAY FROM PILOT-MON , 500
51	Flexible constrnt w/import:	EXCLUSION-SET DAY FROM PILOT-FRI , 500
52	Flexible constrnt w/import:	EXCLUSION-SET DAY FROM SAME-DAY , 400
53	Flexible constrnt w/import:	EXCLUDE TIME FROM EXHAUSTED-GOOD , 200
54	Flexible constrnt w/import:	EXCLUDE TIME FROM EXHAUSTED-BAD , 200
55	Flexible constrnt w/import:	CHOICE-SET VISUAL-1 FROM DEMO , 100
56	Flexible constrnt w/import:	CHOICE-SET VISUAL-1 FROM PILOT , 100
57	Flexible constrnt w/import:	MEET TIME FROM VISUALS , 60
58	Flexible constrnt w/import:	MEET TIME FROM CAB , 60
59	Flexible constrnt w/import:	MEET TIME FROM RT , 60
60	Flexible constrnt w/import:	CHOICE-SET VISUAL-1 FROM SAME-DAY-CAB , 30

```
61 Flexible constrnt w/import:      CHOICE-SET RT FROM WSI , 30
62 Flexible constrnt w/import:      CHOICE-SET VISUAL-1 FROM LOW-DEMAND , 10
63 Flexible constrnt w/import:      CHOICE-SET RT FROM LOW-DEMAND , 10
64 Flexible constrnt w/import:
65 Justification:                   DETERMINED BY ATC PROGRAMMERS BASED ON
                                      INTERVIEWS OF SIMLAB STAFF, 4/89.
```

This is the workhorse procedure for the SIMLAB scheduler. TILE is the name the SIMLAB uses for the commitment of time and equipment for each session, or SHIFT, requested. This procedure combs through the BLOCKs of available time and equipment using the constraints listed. The procedure ALLOCATEs time and equipment to the TILE from the BLOCK that meets all of the hard constraints and gets the best rating on the flexible constraints. (The method of rating candidates according to flexible constraints met is described in the next section.) The lack of a precondition means that this procedure should be used for every tile created. If there is more than one SHIFT per REQUEST and more than one REQUEST, the procedure will be applied iteratively over REQUESTs and SHIFTs of REQUESTs.

Below are two pieces of knowledge that describe constraints mentioned in the knowledge piece 1 procedure. The schemas start by collecting identifying information -- what is constrained, how it is constrained, and a keyword to identify the source of the constraint. The schemas also collect a procedure for how to get a value for the constraint.

Knowledge piece 2 refers to a hard constraint named in the allocation of TILE. It restricts the RTs, or real-time computers, that can be used on a request. If the request needs one or more printer-plotters, only the RTs listed can be assigned to the request (because the printer-plotters are hard-wired to these RTs).

```
Knowledge Piece (2)

 1  Constrained value:          RT OF TILE
 2  Constraint type:            CHOICE-SET
 3  Constraint source:          PRINTER-PLOTTER
 4  Constraint name:            CHOICE-SET RT OF TILE FROM
                                  PRINTER-PLOTTER
 5  Precondition:               PRINTER-PLOTTER OF REQUEST > 0
 6  Procedure:                  LIST
 7  List of values:             RT1 , RT2 , RT3 , RT4 , RT5 , RT7 ,
                                  RT8 , RT9
 8  Justification:              THE PRINTER-PLOTTERS ARE HARD-WIRED TO
                                  THE RT'S LISTED.
```

Knowledge piece 3 represents a constraint that restricts the allocation of a block that would bring a programmer or engineer in early after working late on some other tile the night before. If the programmer or engineer on a request is the same as that on a tile and a block is being considered on the next day, the earliest start time must be 12 hours past the end of the tile (minus a constant to reset the clock for going past midnight). This is used as a heavily weighted preference in the allocation of TILE.

```
Knowledge piece (3)

 1  Constrained value:        START-TIME OF TILE
 2  Constraint type:          MINIMUM
 3  Constraint source:        LATE-EARLY
 4  Constraint name:          MINIMUM START-TIME OF TILE FROM
                                LATE-EARLY
 5  Precondition:             [ [ PROGRAMMER OF REQUEST = PROGRAMMER
                                OF TILE :2 ] OR [ ENGINEER OF REQUEST
                                = ENGINEER OF TILE :2 ] ] AND
                                NEXT-DAY [ BLOCK , TILE :2 ]
 6  Procedure:                CALCULATION
 7  Formula:                  END-TIME OF TILE :2 + 12 - CLOCK-RESET
 8  Justification:            DON'T BRING SOMEONE IN EARLY AFTER
                                WORKING LATE THE NIGHT BEFORE.
```

SILICA runs a number of checks on how the basic knowledge pieces, like the ones just shown, fit together. SILICA checks for obvious omissions -- things that have been mentioned in procedures that may need procedures of their own. For example, the following is one message that would be issued by an omissions check run if knowledge piece 3 alone were entered:

```
I have no procedure for obtaining a value for CLOCK-RESET which is
used in the calculation of MINIMUM START-TIME OF TILE FROM LATE-EARLY.
Do you wish to specify one now? [ SAVE ]:
```

A special kind of omissions checking is provided for objects, such as REQUESTs and TILEs, that have several attributes. Help is provided to make sure the attributes are grouped properly and that when objects are manipulated by procedures, values for all of their attributes are accounted for. These kinds of ommissions checks try to ensure that all the foundational pieces of knowledge are present.

2.2. Explicit Ordering Knowledge

Once the base knowledge pieces are collected, SILICA must determine what decision ordering among pieces should be compiled for the performance system. The bare minimum of control -- actually the lack of explicit control -- would allow the procedures to apply according to data-driven control. SILICA runs a check to make sure that compiling the procedures with data-driven control is possible, and queries the user about potential trouble spots. In cases where procedures are iterative, there are ambiguities about the organization of the iteration that must also be resolved. Finally, there are obvious places where additional strategic knowledge can be used to affect system performance. This section describes the decision-ordering checks and the additional knowledge elicited.

For each piece of knowledge entered, SILICA identifies the dependency relations with other pieces in the knowledge base and stores these links in its permanent representation of the knowledge base. A sample of the network is shown in figure 1. For readability, the figure shows only two of the links derived from knowledge piece 1. (In the complete knowledge base, a link is derived from BLOCK and from each constraint mentioned in the procedure to TILE.) The figure also shows the link from TILE to the MINIMUM START-TIME constraint derived from knowledge piece 3.

SILICA first looks for loops in the dependency network. (To make the example easier to follow,

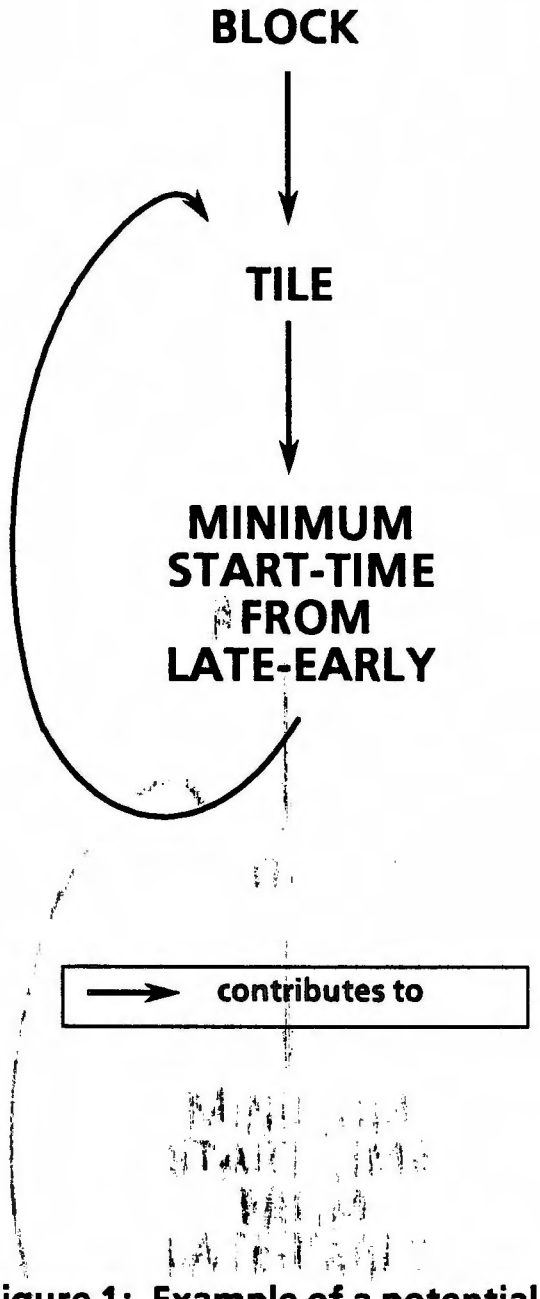


Figure 1: Example of a potential loop in the sample knowledge base.

we will continue to demonstrate the facility for a knowledge base consisting of knowledge pieces 1 and 3.) If TILE is needed to determine the constraint value and the constraint is needed to determine the TILE, there could be an impasse when the generated system runs. Where should the expert system start assigning values? Because both procedures are iterative (they'll be eligible for application at least once for each REQUEST), SILICA first asks whether the constraint that contributes to TILE and the one TILE contributes to are created on different cycles:

```
There is a loop in the knowledge base involving the following chain
of objects:

    1. TILE
    2. MINIMUM START-TIME OF TILE FROM LATE-EARLY

I assume that MINIMUM START-TIME OF TILE FROM LATE-EARLY contributes
to a different TILE than the one that contributed to it.  Is this
what you intended [YES]: <cr>

Which procedure on the loop will apply first on the first
cycle [TILE]: <cr>
```

The user agrees and is then asked where the cycle starts. If the user did in fact intend to state a mutual dependency, SILICA would guide the user in how to use propose-and-revise to handle the loop in the same way that SALT does (see [Marcus 88a]).

Since each of the procedures (the ones represented by knowledge pieces 1 and 3) are iterative, SILICA needs to know how to manage the iteration. SILICA first asks whether to iterate over each procedure before moving on to the next or whether to iterate over longer chains of procedures:

```
Procedures for the following involve REQUEST objects, and the
procedures form a chain:

    1. TILE
    2. MINIMUM START-TIME OF TILE FROM LATE-EARLY

Should each procedure on this chain be applied for every REQUEST
before the next procedure is applied [YES]: NO

The entire chain will be applied to each REQUEST.

Please describe how the REQUEST objects should be ordered...
```

Once the extent of iteration is established, SILICA asks for an important piece of strategy -- how to order the REQUESTS for consideration. The PROCEDURE supplied by the SILICA user below calls for selecting each request for consideration according to properties of the requests that affect the constraints used in TILE allocation:

```
1  Name:                          ACTIVE-REQUEST
2  Precondition:                  NONE
3  Procedure type:                SELECTION
4  Candidate set:                 REQUEST
5  Hard constraint:               NONE
6  Flexible constrnt w/import:    CHOICE-SET DEMO FROM REQUEST , 1000
7  Flexible constrnt w/import:    CHOICE-SET PILOT FROM REQUEST , 100
8  Flexible constrnt w/import:    CHOICE-SET VIEWPOINTS FROM REQUEST , 50
9  Flexible constrnt w/import:    CHOICE-SET CAB FROM REQUEST , 50
10 Flexible constrnt w/import:    DONE
11 Justification:                 DETERMINED BY ATC PROGREMMERS BASED ON
                                    INTERVIEWS OF SIMLAB STAFF, 4/89.
```

Finally, there is one additional piece of knowledge that affects decision ordering in a fairly complex way. This knowledge sets a threshold on the importance of preferences within a given allocation procedure. The threshold for the allocation of TILE shown in knowledge piece 1 is set by the user to be 100. This is effectively an instruction that every flexible constraint weighted above 100 is considered to be very important. The generated performance system should make every effort to satisfy it and warn the user if it is relaxed. Every flexible constraint weighted 100 or less is considered less important and is expected to participate in tradeoffs with other flexible constraints in the less-important category.

These last two kinds of information -- the selection criteria on indexes that regulate iteration and the threshold on importance of flexible constraints -- explicitly determine decision ordering in a way that affects the efficiency and effectiveness of the generated performance system. Those effects are described in more detail in the next section.

3. Variations in Strategy

What makes decision ordering difficult for most design applications is the interdependence among individual decisions. Decisions a designer makes on one part of a design can affect other parts of the design which, in turn, can affect decisions on the first part. For any decision ordering, backtracking may be required to redo work on an earlier part of the design when constraints can't be met on a later part. The goal in ordering the decisions is to find a good solution in a reasonable amount of time. The trick is to avoid early commitments in one part of the design that will lead to unnecessarily suboptimal choices in another part and to make commitments in such a way as to minimize the frequency and extent of backtracking.

3.1. Ordering Iterations of TILE Allocations

For the SIMLAB domain, the quality of the solution and the speed with which it is reached are affected by the order in which the requests are considered. Repetitions should be ordered so that the most difficult-to-satisfy constraints are addressed first. This means ordering the requests according to how constrained they are. Dealing with more constrained requests before less constrained ones makes it more likely that the resources required by each request will be available and unassigned at the time the request is considered. This will reduce the amount of backtracking needed to satisfy hard constraints.

A second factor that enters into ordering the requests is the importance of the constraints associated with each request. In the SIMLAB domain, many constraints are soft; that is, they can be relaxed, and they can enter into tradeoffs with other soft constraints. These constraints

differ in how important they are to satisfy and how much emphasis they should be given when the system makes tradeoffs. In such a case, more important constraints should be addressed earlier because they should have the first chance at the resources. If the problem-solving strategy does not backtrack when a soft constraint cannot be met, then ordering requests according to constraint importance is critical to the quality of the solution. Even if backtracking is invoked when a soft constraint fails to be met, the amount of disruption is less and the efficiency and probability of success are improved if more important constraints are dealt with before less important ones on the first pass.

The two factors -- the difficulty and the importance of satisfying constraints -- can interact. This makes it difficult to specify ways of categorizing requests for ordering their consideration. Because it is difficult to specify in advance, it should be made easy to change as the system is tested on real data. SILICA provides the interface that allows a user to adjust this aspect of strategy easily. The critical knowledge -- the identification of ordering constraints and their importance -- is supplied and easily changed by the SILICA user.

3.2. Threshold on Constraint Importance

Setting a threshold on constraint importance puts a discontinuity on the importance function that is reflected in the way the performance system treats the two categories of constraints. There are computing costs associated with each treatment and judging the importance of satisfying a constraint versus the computing cost of doing so using either method is fairly complex. We'll begin by describing the difference in treatment.

The difference between the treatment of very important and less important flexible constraints is that the very important ones are treated as filters and the less important ones have the action of voters. In SIMLAB's allocation of TILEs, each constraint looks at the available BLOCKs to decide which candidate will be used to create the TILE. We will call a decision a filter if it makes a pass/doesn't-pass decision. A decision is a voter if it adds some weight representing degree of preference for a particular assignment (assuming that the final selection step takes the "pass" candidate with the highest degree of preference).

Any hard constraint must be treated as a filter. Any soft constraint that can enter into a tradeoff with less important soft constraints (because of a cumulative affect of the less important constraints) is a natural as a voter. Any soft constraint that must not enter into tradeoffs with less important constraints is like a removable filter; that is, it's a filter, but if nothing passes it, the performance system takes actions to relax the filter.

The difference between treating a constraint as a filter and treating it as a voter is the difference between committing yourself to a theoretically best alternative and backtracking if necessary versus committing yourself to a lot of bookkeeping to analyze alternatives before selecting one. It makes sense to use the second approach if you know it's unlikely you'll get everything you want and the cost of the extra bookkeeping is less than the cost of all the backtracking. The fact is, it's hard to predict each of these costs in advance and therefore difficult to categorize constraints. It's also difficult to set it once and for all because as the population of requests changes, the likelihood of satisfying each constraint might also change. Once again, SILICA provides the interface to facilitate changes in this aspect of strategy. Through SILICA, the user

can change the threshold, as well as any constraint importances, both to fine-tune the system and keep pace with changing dynamics of the domain.

4. Testing Strategy Variations

This section is a placeholder for a report on work that is in progress. We plan to test selected variations in strategy described in the last section using real data from the Boeing Commercial Airplane Flight Simulation Laboratory. The testing is part of our validation of the SIMLAB scheduler. It has started now and we will be prepared to report on it by the time of the Banff meeting.

5. Current Status

In fall of 1987, we delivered a version of the SIMLAB Scheduler and a version of SILICA that can maintain a portion of the Scheduler knowledge base (that portion which is similar to the knowledge base of VT, an elevator design system supported by SALT [Marcus 88b]). The delivery and use of the system served to verify our assessment of the applicability of SILICA knowledge roles and the assumed problem-solving strategy. Since that time the Scheduler has been in constant production use, and the reduced version of SILICA is run occasionally as the need to update the Scheduler arises because of simulation equipment changes. The Scheduler has reduced the amount of time needed to prepare the schedule, produces higher quality schedules than the unaided human, particularly in human factors terms, and makes the SIMLAB more responsive to last minute changes. The SIMLAB has also added many more pieces of equipment since we first began the project, and the Scheduler has helped them keep pace with the growth in complexity of the scheduling task.

Delivery of the enhanced SILICA described in this paper is scheduled for May 1989. The expected benefit is both a lower cost of maintaining the scheduling system and a better maintained system.

Acknowledgements

Jeff Cline contributed to the coding of the scheduler I/O routines. We are grateful to Dixie Blair, Jeff Cline, Dean Lougheed, and Jerry Smith of the Boeing Commercial Airplane Flight Simulation Laboratory for their help and support throughout the entire SIMLAB project.

Please note that the SILICA examples in this paper are only loosely based on SIMLAB. They were not entered by SIMLAB personnel but were constructed by the authors for demonstration purposes.

References

[Boose 84] Boose, J.
Personal construct theory and the transfer of human expertise.
In *Proceedings of the Fourth National Conference on Artificial Intelligence.* Austin, Texas, 1984.

[Boose 89] Boose, J., and B. Gaines (editors).
The Third Knowledge Acquisition for Knowledge-Based Systems Workshop KAW-88: Working Group Reports.
Knowledge Acquisition for Knowledge-Based Systems Newsletter 1(1):3-4, February, 1989.

[Davis 77] Davis, R., B. Buchanan, and E. Shortliffe.
Production Rules as a Representation for a Knowledge-Based Consultation Program.
Artificial Intelligence 8(1):15-45, 1977.

[Doyle 79] Doyle, J.
A truth maintenance system.
Artificial Intelligence 12(3):231-272, 1979.

[Eshelman 86] Eshelman, L., and J. McDermott.
MOLE: A knowledge acquisition tool that uses its head.
In *Proceedings of the Fifth National Conference on Artificial Intelligence.* Philadelphia, Pennsylvania, 1986.

[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 87] Klinker, G., J. Bentolila, S. Genetet, M. Grimes, and J. McDermott.
KNACK -- Report-driven knowledge acquisition.
International Journal of Man-Machine Studies 26(1):65-79, 1987.

[Marcus 88a] Marcus, S.
SALT: A Knowledge Acquisition Tool for Propose-and-Revise Systems.
In S. Marcus (editor), *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers, Boston, Massachusetts, forthcoming, 1988.

[Marcus 88b] Marcus, S., J. Stout, and J. McDermott.
VT: An expert elevator configurer that uses knowledge-based backtracking.
AI Magazine 9(1):95-112, 1988.

[Marcus 89] Marcus, S.
Understanding Decision Ordering from a Piecemeal Collection of Knowledge.
Knowledge Acquisition , forthcoming, 1989.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Understanding Metaphors by Frustration-Based Learning Method

Masaki Suwa (e-mail suwa%harl86.harl.hitachi.co.jp@uunet.uu.net)
Hiroshi Motoda (e-mail motoda%harl86.harl.hitachi.co.jp@uunet.uu.net)
Advanced Research Laboratory, Hitachi, Ltd.
Kokubunji Tokyo 185, Japan.

Abstract

Understanding a metaphorical sentence means to discover some "general metaphors" which underlie the sentence. In this paper, we claim that Frustration-Based Learning method (FBL method) can be applied to understanding metaphors. FBL method has two main processes; identification of frustrated states and limited-forward reasoning. The first process contributes to discovering where to learn in the whole problem. The latter is to do limited-forward reasoning within the restricted world which has been concerned with each frustrated state in the first task, and it directly contributes to discovering all the pieces of information relevant to the corresponding frustrated state which themselves constitute a general metaphor.

1 INTRODUCTION

A metaphor is so much pervasive in our daily life (Lakoff, 1980) that we cannot perform speech acts or thinking activities without it. A metaphorical expression means to regard a certain concept as something else. Through the act of mapping some features of the latter concept to the former, we can direct our attention toward some features of the former concept which otherwise could not flash into our minds and consequently we can bear a vivid and exact description of the concept in our minds (Yamanashi, 1988).

Norvig suggested in his works on inferences about text understanding (Norvig, 1986; Norvig, 1987) that a text understander can process metaphorical sentences in the same way as normal sentences if it is in advance provided with *view-links* between two concepts as parts of domain knowledge. His proposal is reasonable considering that a metaphor works on mapping relations between two concepts with different semantics (Lakoff, 1980). However, it is almost impractical to provide in advance a text understander with a variety of view-links (in this paper they are called viewing structures) which help to effectively cover as many patterns of metaphors as possible.

The approach in the researches on metaphors has been to produce new view-links for understanding a new metaphorical sentence by finding analogies between the new sentence and the old metaphorical one already understood and by mapping some view-links of the old one onto the new one (Gentner, 1983; Martin, 1987; Martin, 1987; Winston, 1980). However, an analogy-based mapping does not lead to deep comprehension of why the new sentence can be a metaphorical expression.

The greatest significance of Lakoff's work is to recognize existence of general metaphor in our daily life and to claim that a metaphor appearing in a specific metaphorical sentence should be an instantiation of the general metaphor. A general metaphor is made up of lots of view-links between features or entities included in two concepts. For example, a metaphor in such a sentence as "Your argument needs more foundation" is an instantiation of general metaphor "Argument is a kind of building" [1]. We can say that deep comprehension of a metaphorical sentence can be attained only when a metaphorical sentence is associated with the general metaphor which underlies the sentence and consists of much larger viewing structures than those which would be acquired by analogy-based mapping methods.

Carbonell (Carbonell, 1980) made a clear specification about what kinds of information general metaphor should have and proposed a method for understanding new metaphors by use of such information. However, he did not mention a learning system to process metaphorical sentences.

The aim of this paper is to show that Frustration-Based Learning system (FBL system) (Suwa and Motoda, 1989; Suwa, 1989) has an ability to process a new metaphor and to acquire the viewing structures of the general metaphor. FBL method has first been proposed in the domain of auxiliary-line problems in elementary geometry for acquiring effective strategic knowledge for solving target problems. In the second section, the general framework of FBL is described precisely. In the third section, an application of FBL method to understanding new metaphors is illustrated by an example. In the fourth section, what significance FBL method has in understanding metaphors is discussed in the light of Carbonell's specifications about general metaphor. And in the fifth section, FBL is compared with the other learning methods.

2 GENERALIZED FRUSTRATION-BASED LEARNE

1. **Representation of a problem world**
   In FBL systems, a problem world which is dealt with is represented as an assembly of individuals and relations. Individuals are real or abstract entities existing in the world, while relations are the features of the individuals, or constraining information among individuals, or interrelations among relations. An individual is represented as an atom, and a relation is represented as a literal in predicate logic.

2. **Knowledge**

   - **Domain knowledge** Domain knowledge consists of the one about individuals and the one about relations. The former knowledge expresses what an individual is like, specifically what other individuals constitute the world concerning the individual and what relations are accompanied with the world. The latter knowledge expresses what individuals should be involved in a relation and what interrelations hold among some relations.
   - **Knowledge about inferences** In FBL as well as in all the other existing problem-solving systems, *problem-solving* is regarded as forming a sequence of *local inference actions*. So we have to define the type of *local inference action* in every problem domain. Here, local inference action means an operation which

[1] This example is taken from (Lakoff, 1980).

relates one set of relational information to another set of relational information for the purpose of producing new problem elements from the existing ones. Further, we need to provide a few *inference rules* which are very specific to the domain itself. The goal and given conditions of a problem, and subgoals and conditions which are produced during the process of a problem-solving are called *problem elements*. The aim of a problem-solving is to produce in the *problem space* the problem element which is same as the goal.

3. **Problem-solving processes**

   - **Identification of frustration elements** In FBL system, those states in which no local inference actions can be further performed using a certain problem element due to the lack of some other problem elements in a problem space are called as *frustrated states*, and the problem element which remains still unused is called a *frustration element*. Identifying several frustration elements during a problem-solving process is the first important task that FBL system does, and this task is significant because it will be a motivation for learning.
   - **Resolution of frustrated states** FBL system must resolve these frustrated states in order to solve problems. Resolution of each frustrated state means to make the frustration element usable and consequently to bring about the corresponding local inference, by adding appropriate problem elements into a problem space either according to a user's suggestion or by exhaustive search, where that local inference must be a part of a correct sequence of actions which connects the given conditions to the goal. There may be some unresolved frustrated states, but we will focus only on the resolved ones in the following processes.

4. **Learning processes**
   A learning process consists of the following two tasks, which should be performed for each of the frustrated states whose resolution has made a contribution to the problem solving process.

   - **Restrictions of a world** This task is 1) to extract those individuals from the pieces of relational information which constitute the local inference action corresponding to each frustrated state, and the other individuals which are referred to in the body of knowledge concerning each of those extracted individuals, and then 2) to establish the world which is composed of only those extracted individuals in 1) as a restricted world.
   - **Limited forward reasonings** We mean by "limited" reasonings within the restricted world the ones which are performed using the pieces of relational information holding only on those extracted individuals. This task is to obtain all the pieces of information which hold in the restricted world, initiated by the pieces of relational information that constitute the complete proof tree of a problem.

5. **FBL-knowledge**
   FBL system generalizes all the pieces of relational information obtained in the learning process into the following style of FBL-knowledge.

IF a frustration element

THEN

an assembly of the pieces of relational information which are obtained in the limited forward reasoning process corresponding to the element.

The method of generalization is basically the same as that in Explanation-Based Learning method (Mitchell et al, 1986; Dejong et al, 1986): to generalize each piece of information by reserving the constraints in the domain knowledge which has been made use of in producing the piece of information.

This is an associative knowledge which tells "If you are faced with a frustration element which matches to its IF part, then find out (or create, or recall) in the problem space an object (a concept) which is defined by all the pieces of information in its THEN part". FBL is based on an idea that what should be learned after experiencing a problem is some pairs of a frustrated state and a body of relevant information to it. And we insist that the pieces of information obtained during this "limited reasonings" are truly relevant to each frustration element.

3 AN APPLICATION OF FBL TO METAPHORS

3.1 Representation of a world of metaphor

A world of natural language is made up of individuals and relational information. Nouns in sentences correspond to individuals and other parts of speech eg. verbs, adjective, prepositions correspond to relational information. A piece of relational information is represented in predicate logic as follows;

$$rel(R, [Arg_1, Arg_2, \ldots, Arg_n])$$

where

R = the name of a relation,

Arg_i $(1 \leq i \leq n)$ = relational information or an individual.

3.2 Knowledge in the domain of text understandings

3.2.1 Domain knowledge

- **Knowledge about individuals** (see Fig.1)
  The knowledge about an individual consists of **1)** other individuals associated with the individual, and **2)** relational information associated with the individual, such as a hierarchical relation (hyponymy), for example *a-kind-of* or *instance*, between the individual and its superordinate concept, relations among individuals within this body of knowledge, actions or states associated with these individuals, higher relations among relations (eg. causation, pre-condition, significance, ... etc.). Such a body of knowledge expresses a world in the common sense which is asssociated with the individual.

- **Knowledge about relations** (see Fig.2)
  In this paper, only verbs, prepositions and adjectives which are accompanied with the

Individuals

team1, team2, fan, ampire, manager, ground, baseball-gear. player, sponsor, strategy. position, runner, batter, pitcher, excitement, ...

Relations

- **hyponymy**
  rel(a-kind-of, [baseball-game, event])
- **relations between individuals**
  rel(opponent, [team1, team2]), rel(agent, [team, baseball-game]),
  rel(a-kind-of, [pitcher, position]), rel(member-of, [player, team]), ...
- **actions and states**
  rel(control, [manager, baseball-game]), rel(watch. [fan, baseball-game]), rel(defeat, [team1,team2]), rel(win, [team, baseball-game]), rel(attack, [team1, team2]),
  rel(defense-against, [team1, team2]), rel(give, [team. fan, excitement]), ...
- **higher relations**
  rel(purpose,[team1, rel(opponent, [team1, team2]), baseball-game,
  rel(defeat, [team1, team2])]),
  rel(has-significance, [baseball-game, rel(give, [team. fan, excitement])]),
  rel(cause, [rel(defeat, [team1, team2]), rel(support, [fan, team2]),
  rel(sad, [fan])]),
  rel(one-of-stages, [rel(attack, [team1, team2]), baseball-game]), ...

Figure 1: **Domain knowledge about an individual(eg. Baseball game)**

verb *be* are processed. The knowledge about a relation expresses several kinds of constraints among arguments which the relation posseses. Such knowledge is represented using a higher relation *constraint*. The first argument of a *constraint* relation is the target relation and a list of all the rest arguments is a set of conjunctive constraints.

3.2.2 Knowledge about inferences

In this paper, literal inputs of problems, problem elements, the objective of inferences and finally a local inference action are specified as follows;

- **Literal inputs**
  A text understander is given as inputs an assembly of finite pieces of relational information which constitute a text.
- **Problem elements**
  Those pieces of relational information in the literal inputs correspond to given subgoals and all the pieces of relational information which exist in the domain knowledge or are produced during problem-solving processes correspond to problem conditions. Notice that problem conditions are not given as initial conditions of the problem in this

```
rel(constraint, [ rel(crush, [A, B]),
                  rel(a-kind-of, [A, person]), rel(a-kind-of, [B. objects])]).
rel(constraint, [ rel(crush, [A, B]),
                  rel(a-kind-of, [A, army]), rel(a-kind-of, [B. army])]).
                  ⋮
```

Figure 2: **Domain knowledge about a relation (eg. a verb "crush")**

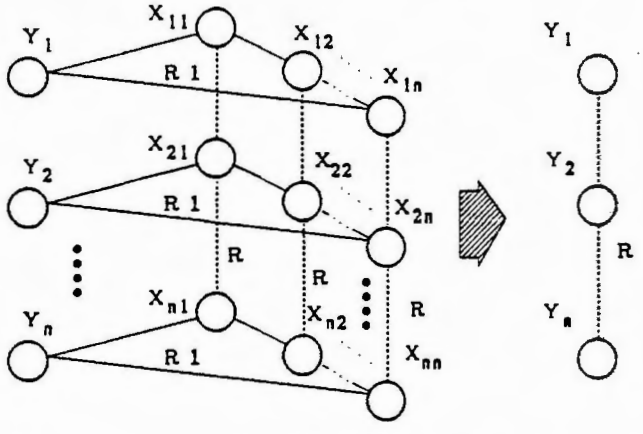


Figure 3: **A local inference action in the domain of text understanding**

domain, unlike in the domain of elementary geometry and that a text understander itself must find necessary problem conditions out of enormous amount of domain knowledge during problem solving processes.

- **The objective of inferences**
  The objective of inferences is to verify all the pieces of literal inputs, which are the subgoals of the problem, by finding out a sequence of local inference actions by use of domain knowledge.

- **Local inference actions**
  The basic style of local inference actions is illustrated in Fig.3. It is a kind of meta-knowledge which says that a new piece of relational information, $rel(R, [Y_1, Y_2, \ldots, Y_n])$, can be produced in the problem space if all the following requirements are satisfied;

  - $rel(R, [X_{11}, X_{21}, \ldots, X_{n1}])$, $rel(R, [X_{12}, X_{22}, \ldots, X_{n2}])$. $\ldots$, $rel(R, [X_{1n}, X_{2n}, \ldots, X_{nn}])$ should exist in the problem space or in the body of domain knowledge, and
  - $rel(R1, [Y_1, X_{11}, X_{12}, \ldots, X_{1n}])$, $rel(R1, [Y_2, X_{21}, X_{22}, \ldots, X_{2n}])$, $\ldots$, $rel(R1, [Y_n, X_{n1}, X_{n2}, \ldots, X_{nn}])$, should exist in a body of domain knowledge.

For example, a piece of relational information, $rel(defeat, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees])$, is verified by the following local inference action;

$$rel(defeat, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees]) :-$$
$$rel(instance, [Sanfrancisco\text{-}Giants, Baseball\text{-}team]),$$

$rel(instance, [NewYork\text{-}Yankees, Baseball\text{-}team])$,
$rel(defeat, [Baseball\text{-}team, Baseball\text{-}team])$.

3.3 Verifying processes of texts

We show, as an example, the problem-solving processes of a metaphorical sentence, "The fans were satisfied with the last night's baseball-game where Sanfrancisco-Giants crushed New York-Yankees".

3.3.1 Literal inputs

Such subgoals as $rel(crush, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees])$ and $rel(satisfied_with, [fan, baseball\text{-}game])$ are given as literal inputs.

3.3.2 Inferences

The latter subgoal is verified by the following local inference action;

$$rel(satisfied_with, [fan, baseball\text{-}game]) :-$$
$$rel(a\text{-}kind\text{-}of, [fan, person]),$$
$$rel(a\text{-}kind\text{-}of, [baseball\text{-}game, event]),$$
$$rel(satisfied_with, [person, event]).$$

Each piece of relational information in the body-part of the above local action is found in the knowledge base about *fan*, *baseball-game* and *satisfied_with* respectively.

Those pieces of relational information which do not include any metaphorical expression are basically verified by one step of local inference action, when the $R1$ relation in Fig.3 is instantiated only by *a-kind-of* or *instance*. On the contrary, a piece of information which *does* include a metaphor cannot be verified in such a way.

3.3.3 Identification of frustrated states

Suppose a situation where an action *crush* does not exist in the knowledge base "*baseball-game*" and at the same time *baseball-game* is not permitted to be an object of the verb *crush* in the knowledge base "*crush*". In this case, such a local inference action as in the above example is not performed for verifying $rel(crush, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees])$ (see Fig.4). Carbonell called this state *semantic case-constraint violation*. Such a state means an existence of metaphor (Carbonell, 1980; Yamanashi, 1988). In this paper, this state is called frustrated state and the subgoal, $rel(crush, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees])$, is identified as a frustration element.

The transformation rule shown in Fig.5 is required for processing metaphorical expressions. The rule is the first example of the "inference rule" specific to the domain of metaphor (see section 2) and it is justified by the common sense knowledge that a metaphorical expression should be interpreted as a set of viewing structure between two situations. When this rule is used in a backward direction for the purpose of verifying a subgoal which includes a metaphorical expression, the body-part of this rule is then called, resulting in recursive executions of local inference actions.

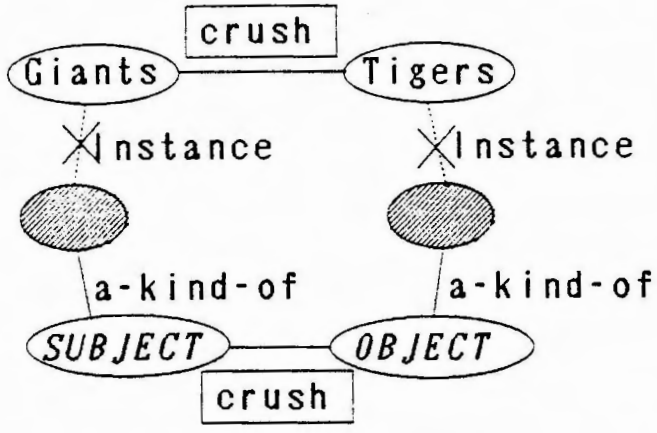


Figure 4: **Semantic case-constraint violation**

$rel(R, [X_1, X_2, \ldots, X_n]) : -$
$rel(viewing, [rel(R, Arg_list1), rel(RNEW, Arg_list2)]).$
where
X_i $(1 \leq i \leq n)$ is a word in the text,

on condition that
each X_i is necessarily included in either Arg_list1 or Arg_list2.

Figure 5: **Transformation rule for interpreting a metaphorical expression**

3.3.4 Resolution of frustrated state

Each of these recursive executions will generate a new subgoal. such as $rel(viewing, [?, ?])$. However, unless there exists any viewing relation in the body of knowledge base or it is appended into the problem space, the subgoal as a frustration element cannot be verified. In case no viewing relation exists in the body of knowledge base, a new viewing relation must be produced as follows; a word, which is located as a certain argument of *crush* either in the knowledge base about *crush* or in the knowledge base about other words where *crush* appears, is to be connected by a view-link with the word which appears in the same location of argument in the present texts.

For example, a piece of relational information, $rel(viewing. [baseball\text{-}team, army])$, is appended as a hypothesis to a problem space. Then, the above mentioned subgoal is verified through several steps of local inference actions shown in Fig.6, resulting in the resolution of the frustrated state which corresponds to a frustration subgoal. $rel(crush, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees])$. In this process, the following transformation rule is used; This rule is the second example of the inference rule specific to the domain of metaphor.

$rel(viewing, [rel(R, [X_1, \ldots, X_n]), rel(R, [Y_1, \ldots, Y_n])])$
$\leftrightarrow rel(viewing, [X_1, Y_1]), \ldots, rel(viewing, [X_n, Y_n]).$

In general, there are several kinds of viewing relation to be appended as a hypothesis. Out of these, the one which contributes to the verification has to be selected by exhaustive

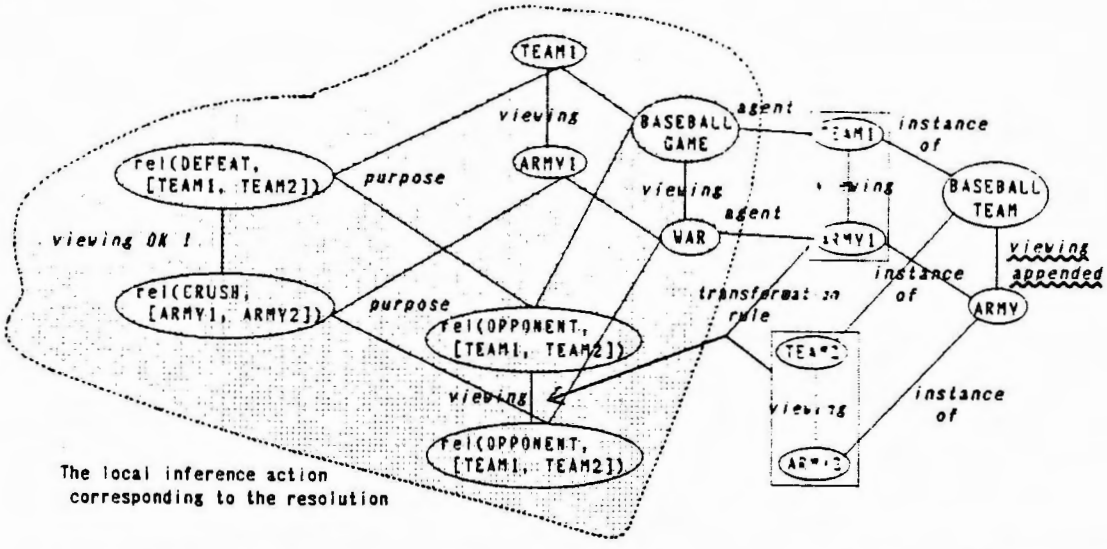


Figure 6: **The local inference actions contributing to resolution of a frustrated state**

search [2].

3.4 Learning processes

The following two tasks are performed for a frustration element $rel(crush, [Sanfrancisco\text{-}Giants, NewYork\text{-}Yankees])$.

3.4.1 Restriction of a world

The local inference action corresponding to the resolution of this frustrated state is illustrated in Fig.6. All the individuals included in this action (baseball-team, army, baseball-game, war) and all the individuals which exist in the knowledge base about each of the above individuals are extracted. Here, we suppose a restricted world which consists of these extracted individuals as well as the pieces of relational information composed only of those individuals.

3.4.2 Limited forward reasoning within the world

The viewing relations which at present hold in the restricted world are
$rel(viewing, [team1, army1])$, $rel(viewing, [team2, army2])$,
$rel(viewing, [baseball\text{-}game, war])$,
$rel(viewing, [rel(defeat, [team1, team2]), rel(crush, [army1, army2])])$.

In this task, the system automatically finds all the pieces of viewing relation which are composed only of the above individuals, by taking successive local inference actions in a forward direction by use of the above viewing relations as initial conditions.

An example of limited forward reasoning process is shown in Fig.7. The transformation rule is utilized twice during this process. In a situation, for example, where an individual *owner-corporation* does not exist in the knowledge base about baseball-game, which means

[2]Sometimes, it is appended by a user's suggestion.

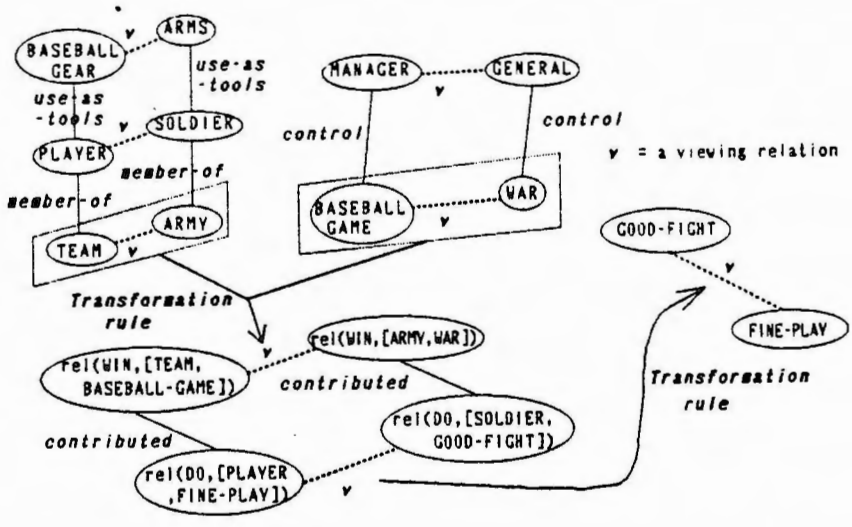


Figure 7: **An example of limited forward reasoning within the restricted world**

that a text understander believes that a baseball-game is not associated with an owner-corporation, then the following inference action is not executed because it is beyond the scope of the restricted world;

$$rel(viewing, [owner\text{-}corporation, nation]) :-$$
$$rel(own, [nation, army]),$$
$$rel(own, [owner\text{-}corporation, baseball\text{-}team]),$$
$$rel(viewing, [baseball\text{-}team, army]).$$

3.5 Acquired knowledge

The following knowledge is acquired;

IF

there exists a subgoal element
$rel(crush, [baseball\text{-}team, baseball\text{-}team])$

THEN

append into a problem space all the **viewing relations** (eg. between war and baseball) which have been obtained in the limited forward reasonings.

The knowledge which the present frustration-based learner acquires in the domain of text understanding does not seem to have high applicability to future problems, because of the specificity of its IF-part to a particular sentence. One way of relaxing this specificity is to permit a matching between the word in its IF-part and the one which exists in the knowledge base where the former word does.

When the FBL-system encounters a new metaphorical sentence which includes the word of "Baseball" world and the word of "War" world in the same subgoal to be verified, it can find a matching between the subgoal and the IF-part of the above acquired knowledge and can come to process the sentence by making use of the viewing relations in the THEN-part.

4 DISCUSSIONS

4.1 Limited forward reasonings

The first important concept of FBL method is the limited forward reasoning within a restricted world. Here, we will explain its significance in two examples.

One is about "implicit intention component" which Carbonell specifies (Carbonell, 1980) as one of the most indispensable information of a general metaphor. The component encodes the reason why this metaphor is typically chosen by a writer or a speaker. When a writer says "John is completely crazy about Mary", which is an instantiation of the general metaphor "love-is-madness", instead of saying "John loves Mary very much", he would intend to express that John may exhibit impulsive or uncharacteristic behavior and that his present state of mind may be less permanent than in the latter case. Therefore, the general metaphor "love-is-madness" must contain the information suggesting that the love may be impulsive and less permanent. Yamanashi also points out that understanding metaphors is accompanied with some new discovery, which can be brought about by the implicit intention component of a general metaphor.

Let's go back to our example. In the limited forward reasoning in our sentence "Baseball team A crushed Baseball team B", the following causal information will be discovered, "If a baseball team is defeated, players of the team will be dead-like", from a local inference using 1) a viewing relation between $rel(crush, [army1, army2])$ and $rel(defeat, [baseball\text{-}team, baseball\text{-}team])$, and 2) a viewing relation between *soldier* and *player* and 3) a causal knowledge in the world of war, "if an army is crushed in a war, soldiers of the army will be dead". This may be a kind of implicit intentional information which should be learned after recognizing this sentence as an instantiation of the general metaphor "Baseball is a kind of war". This is the first example which shows the limited forward reasoning contributes to a new discovery accompanied with understanding a new metaphor.

The second example shows that the limited forward reasoning can produce knowledge which will be helpful to understanding the other sentence which is another instantiation of the same general metaphor. For example, a new sentence "All the soldiers of Redsocks ... ", which belongs to "Baseball is a kind of war", can be understood after learning our sentence because the subgoal, $rel(of, [soldier, Redsocks])$, can be matched by relaxed matching to the IF-part of the acquired knowledge in Section 3 and thus our system can come to use a viewing relation between *soldier* and *baseball player* which is included in the THEN-part of the knowledge.

We can say that what the system acquires by the limited forward reasoning is an assembly of information which underlies the processed metaphorical sentence.

What must be stressed on here is that restricting a world is a requisite premise for performing this reasoning. The reason is; if there is no restriction, inference actions may continue infinitely, such as

$$rel(viewing, [baseball\text{-}team, army]) \rightarrow$$
$$rel(viewing, [owner\text{-}corporation, nation]) \rightarrow$$
$$rel(viewing, [owner, president]) \rightarrow$$
$$rel(viewing, [rel(run, [owner, oner\text{-}corporation]),$$
$$rel(govern, [president, nation])]) \rightarrow \ldots$$

4.2 Identification of frustration elements

The other important concept of FBL method is identification of frustration elements. It has two roles; one is that identification of a frustration element results in giving a criterion of restricting the world within which limited forward reasoning is to be performed (the world is composed of only those individuals which are included in the local inference action corresponding to the frustration element). The other role is to discover targets to be learned in a problem-solving process. In case several semantic case-constraint violations are identified, a structure of viewing relations for each violation is acquired. This act corresponds to understanding the metaphorical text as a composition of these underlying structures.

Each of the acquired viewing structures is regarded as truly relevant to the ways of resolving each frustrated state of a problem. Therefore, if we admit that something like knacks for understanding metaphors is hidden in these frustrated states, we can conclude that FBL method is effective enough to learn these knacks.

5 LEVELS OF UNDERSTANDING

Here, we compare frustration-based learning method with some other learning methods in terms of *levels of understanding.* Three levels of understanding are considered; explanation forming level, solving-processes chunking level, and finally underlying-information discovering level.

The first level of understanding is explanation forming level. Those explanation-based systems which are made to learn classical concept definitions (eg. a concept of cup, seen in (Mitchell et al, 1986; Dejong et al, 1986)) belong to this class: in order to learn a certain feature (target-concept) of a target object, they try to explain why the object is gifted with the feature. Making an explanation can be regarded as a kind of problem-solving, the act of constructing a proof-tree for verifying the feature. This explanation process is significant in that a formation of the process itself contributes to selecting out only the relevant information to the target feature from enormous amount of information concerning the object. What the target feature of an object has been derived from is understood through this explanation process(see Fig.8).

On the other hand, in the domain where the aim is not only to solve problems but also to acquire efficient ways of reasoning and solving, a deeper understanding than the first level is required for attainments of intelligent performances. The second level of understanding is the one obtained by chunking all the lower problem-solving processes under the subgoals which are selected out of the whole problem-solving traces. Those systems which learn macro-operators or control knowledge (Cheng et al, 1986; Laird et al, 1987; Minton, 1985; Minton et al, 1987, Minton and Etzioni, 1987) are categorized in this class. Macro-operator learners extract in an exhaustive manner or in a selective manner (such as PRODIGY) several parts out of the whole problem-solving process within a problem space (see Fig.9). SOAR establishes a whole problem-solving process as composed of multi-layered thinking processes the highest layer of which is a problem space level and learns control knowledge to determine actions in the problem space level by chunking all the thinking processes in the lower level (see Fig.10). In these systems, to recognize by extracting several partial tree-structures of problem-solving traces how a problem has been solved is regarded as an understanding of the problem, because they are based on the concept that effective strategies worth learning are included in the problem-solving traces themselves of the target problem.

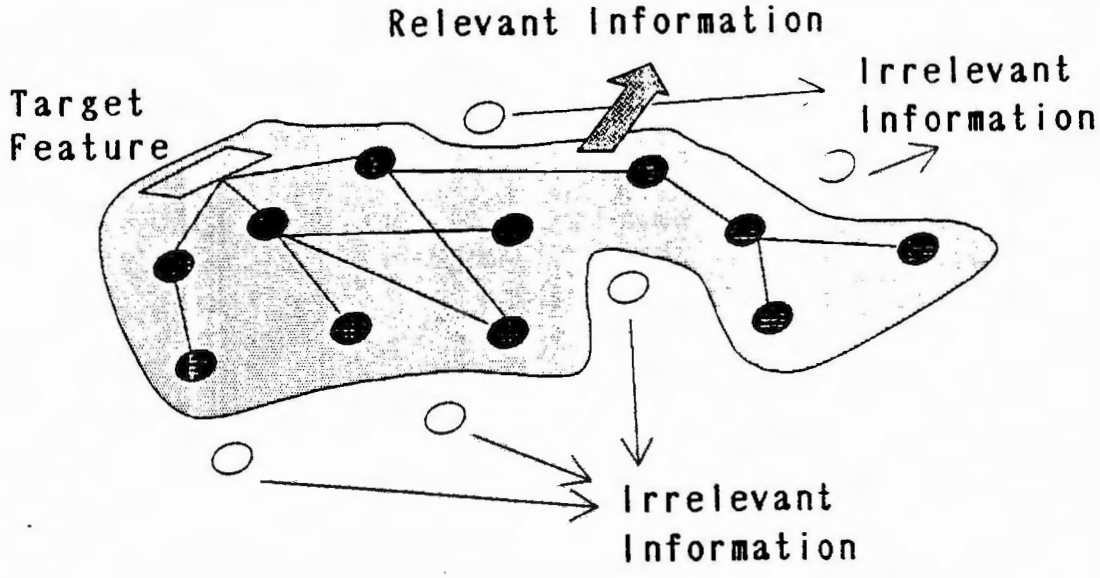


Figure 8: **Understanding by formation of an explanation**

Here, a naive question arises to us. We understand that the knowledge learned by these systems in the second class is a kind of control knowledge which is justified by the verifying processes of the problem and can be used for proving some target-concepts. Is it possible to acquire control knowledge that has been gained preferencially through frequent reinforcement in the past and can be used to control association *without* the *a priori* justification that has characterized the previous works, eg. EBL ? An evidence of the existence of such a knowledge can be found in the domain of auxiliary-line problems in elementary geometry (Suwa and Motoda, 1989; Suwa, 1989). The knowledge learned by those learning systems which are categorized in the first and second class may not be sufficient for controlling associations, because these systems restrict the target of learning to the problem-solving traces of a problem.

It will be discussed that the third level of understanding is attained by a learning system like frustration-based learner. It determines where to learn by identifying *frustrated states* and then collects all the relevant pieces of information to each frustrated state by *limited forward reasonings* which are performed beyond the scope of the problem-solving traces and yet within the world restricted in accordance with the specified criterion when each frustrated state has been resolved. FBL-knowledge is a pair of a frustration element and an assembly of information relevant to it and is sufficient for controlling association, because this assembly contains much information which a learner did not make use of in constructing the problem-solving traces of the past problem but which human beings may implicitly associate the problem with. From the above discussions, we conclude that FBL-knowledge is the one which underlies and structurizes the whole problem-solving process of the target problem (see Fig.11) and that to discover such an underlying assembly of information is the third level of understanding.

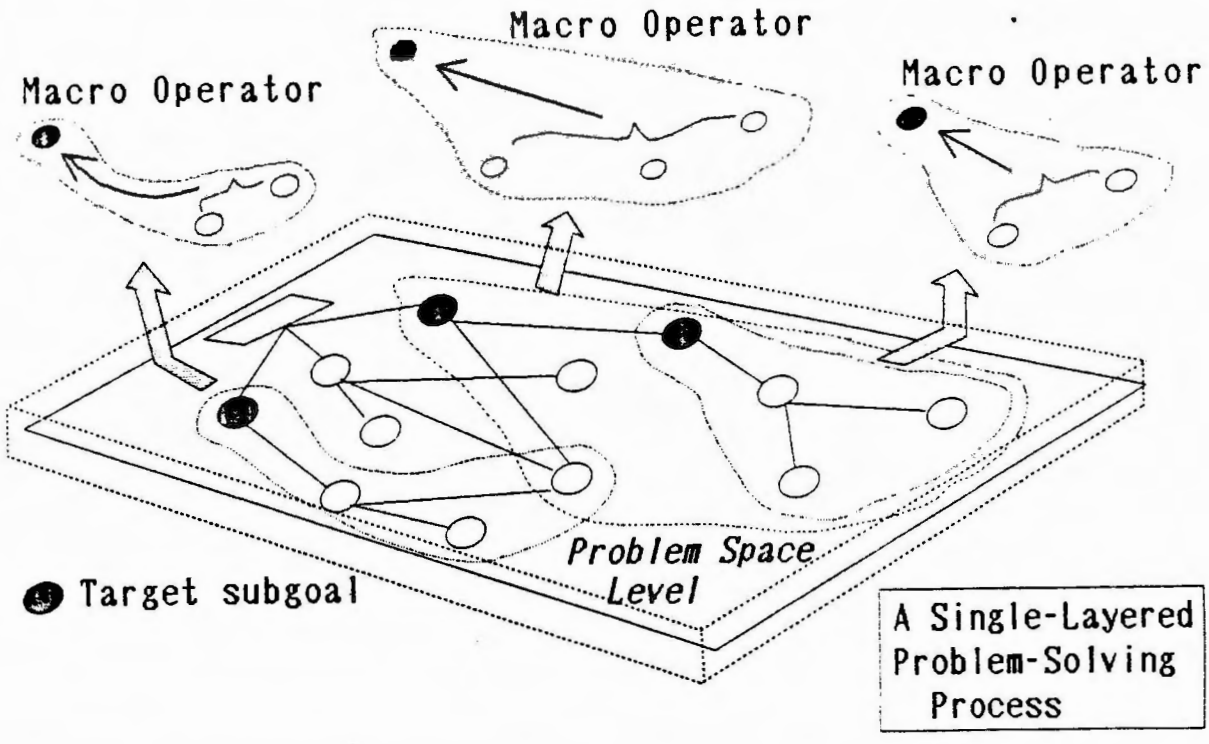


Figure 9: **Understanding by chunking subtrees of a single-layered problem-solving process**

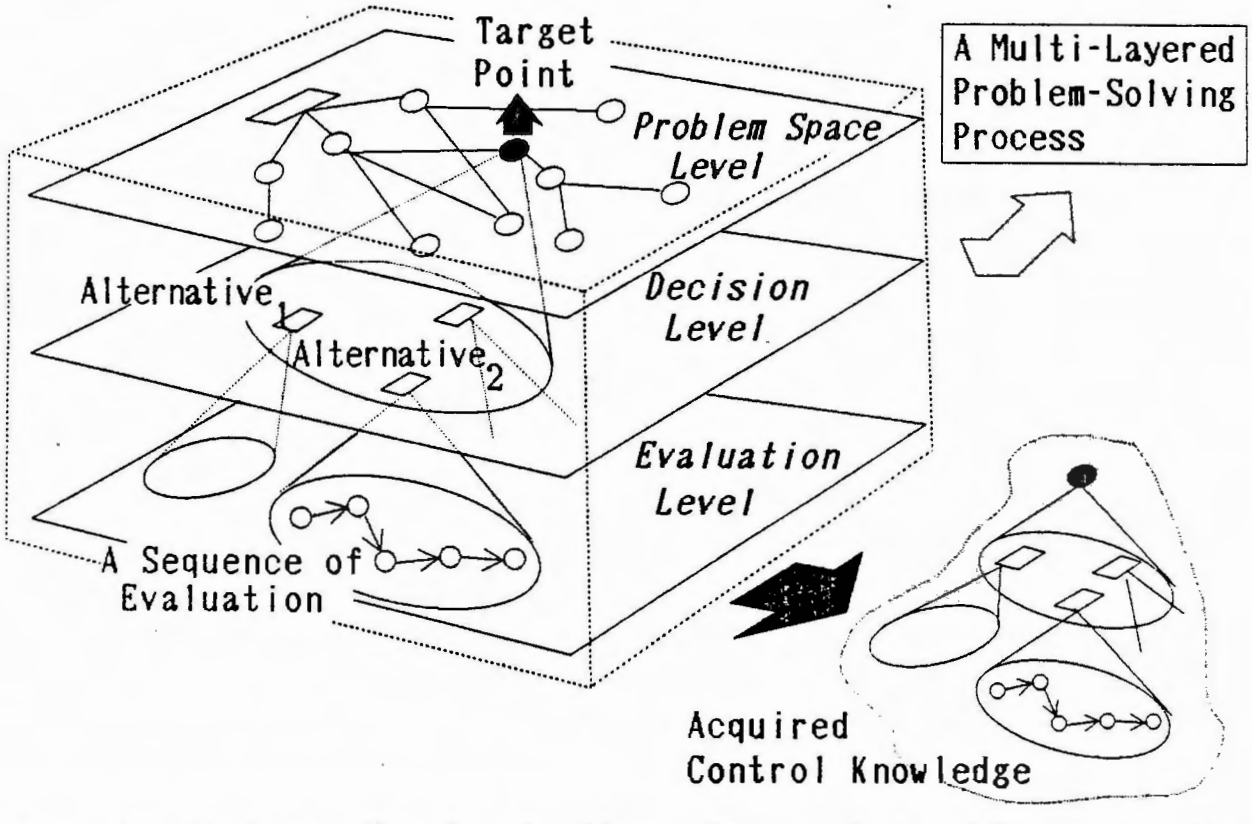


Figure 10: **Understanding by chunking subtrees of a multi-layered problem-solving process**

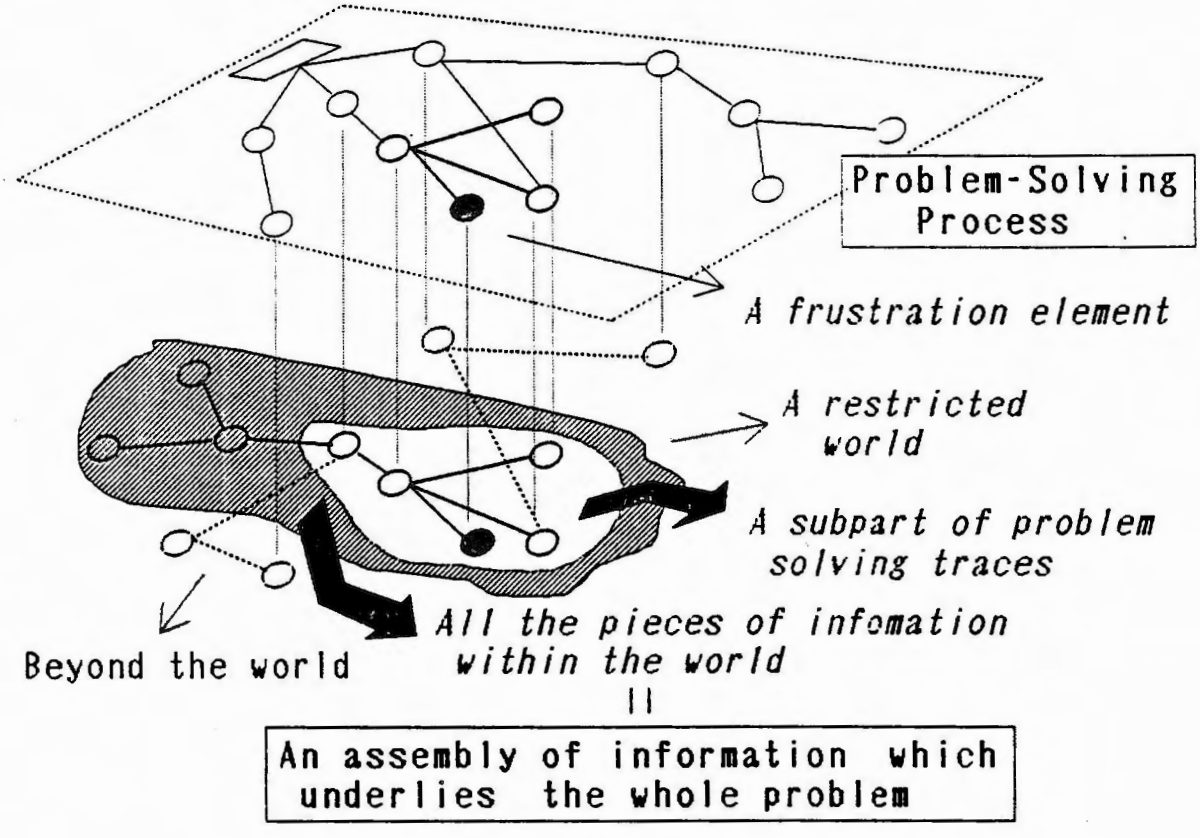


Figure 11: **Understanding by discovering underlying information**

6 CONCLUSION

Frustration-Based Learning method provides the way of establishing frustrated states during problem-solving processes as motive forces to learning and then discovering the body of information which underlies the problem-solving processes. It learns a kind of associative knowledge which is a pair of a frustration element and an assembly of information relevant to it. We can say that learning by FBL-method means understanding the whole problem as a composition of those assemblies of underlying information.

FBL method has two important concepts; identifications of frustrated states and limited reasonings within a restricted world. Performing such reasonings within the restricted world determined in resolving some frustrated states will contribute to discovering the underlying information.

When this learning method is applied to processing a metaphorical sentence, several general metaphors can be acquired in correspondence to several frustrated states, resulting in a deep comprehension of the processed sentence.

REFERENCES

Carbonell, J. G. (1980). Metaphor - A key to extensible semantic analysis, In *Proceedings of the Third Annual Conference of the Cognitive Science Society*, pages 292-295.

Cheng, P. W., and Carbonell, J. G. (1986). The FERMI System: Inducing Iterative Macro-Operators from Experience, In *Proceedings of AAAI-86*, pages 490-495.

Dejong, G., and Mooney, R. (1986). Explanation-Based Learning: An Alternative View, *Machine Learning*, Vol. 1, No. 2.

Gentner, D. (1983). Structure Mapping: A theoretical framework for analogy. *Cognitive Science*, Vol. 7, No. 2.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: An Architecture for General Intelligence, *Artificial Intelligence*, Vol. 33, No. 1.

Lakoff, G., and Johnson, M. (1980). *Metaphors we live by.* University of Chicago Press.

Martin, J. (1987) The acquisition of polysemy, In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 198-204.

Martin, J. (1987). Understanding new metaphors, In *Proceedings of IJCAI-87*, pages 137-139.

Minton, S. (1985). Selectively Generalizing Plans for Problem-Solving, In *Proceedings of IJCAI-85*, pages 596-599.

Minton, S., and Carbonell, J. G. (1987). Strategies for Learning Search Control Rules: An Explanation-Based Approach, In *Proceedings of IJCAI-87*, pages 228-235.

Minton, S., Carbonell, J. G., Etzioni, O., Knoblock, C. A. and Kuokka, D. R. (1987). Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System, In *Proceedings of the 4-th International Workshop on MACHINE LEARNING*, pages 122-133.

Mitchell, T. M., Keller, R. M. and Kedar-Cabelli, S. T. (1986). Explanation-Based Generalization: A Unifying View, *Machine Learning*, Vol. 1, No. 1.

Norvig, P. (1987). Inference in text understanding, In *Proceedings of AAAI-87*, pages 561-565.

Norvig, P. (1986). *A unified theory of inference for text understanding.* PhD thesis, University of California, Berkeley, CA.

Suwa, M., and Motoda, H. (1989). Acquisition of Associative Knowledge by the Frustration-Based Learning Method in an Auxiliary-line Problem, In *Knowledge Acquisition*, Vol. 1, No. 1.

Suwa, M. (1989). An Approach to Cognitive Modeling on Understanding and Learning Processes, PhD thesis, University of Tokyo, Japan.

Winston, P. (1980). Learning and reasoning by analogy. *Communications of ACM.*, Vol. 23, No. 12.

Yamanashi, M. (1988). *Metaphors and Understandings*(in Japanese). University of Tokyo Press.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

Knowledge Acquisition by Abductive and Inductive Explanation

Hirokazu Taki

ICOT Research Center
Institute for New Generation Computer Technology,
4-28, Mita 1-Chome, Minato-ku, Tokyo 108 Japan
Phone: +81-3-456-3192, Csnet: htaki%icot.jp@relay.cs.net

ABSTRACT

This paper describes an explanation-based learning system under incomplete knowledge. We propose an explanation method using abduction and induction. If the system fails to explain a goal and to explain examples, it makes a hypothesis by abduction. In this way, there are many candidate hypotheses. The system uses a few criteria to eliminate candidates. Selection of hypotheses that are in a consistent explanation is explained by induction using a background theory and has minimal generation cost. After explanation generation, the explanation tree is generalized according to operationality criteria. As a result, it generates assumptive macro knowledge.

1. INTRODUCTION

Knowledge acquisition bottleneck is one of most difficult problems in the building of knowledge bases of expert systems. There are two ways to build knowledge bases. The first way is interactive knowledge acquisition: a human

expert is interviewed to extract his knowledge. The other way is knowledge building by understanding observed information. One way to understand something is to try to explain it. EBL (Explanation-based learning) (Mitchell, Mahadevan and Steinberg, 1985; Mitchell, Keller, and Keder-Cabelli, 1986) is a learning method which uses explanation of examples. However, EBL makes only macro knowledge for effective inference. In order to use EBL, a complete domain theory must be prepared before learning. Normally, however, it is difficult to prepare complete knowledge before learning. Therefore, a learning system fails to explain examples. This situation is a trigger which makes other learning occur. Impasse situations of chunking in SOAR (Laird, Rosenbloom and Newell, 1986) and frustration situations of FBL (Suwa and Motoda, 1989) are like failure situations of explanation. Abductive explanation is a kind of explanation method which uses hypotheses. In this explanation, a hypothesis is made by abduction (Charniak and McDermott, 1986). A hypothesis is new knowledge if it does not derive inconsistency. There is a shortcoming in abduction, which generates many candidates as hypotheses to explain examples. It is important to select meaningful hypotheses from all hypotheses by abduction. This paper introduces a hypothesis selection method which considers the operation cost of generalization and specialization of a domain theory. In this method, a hypothesis is selected syntactically using the following criteria.

(1) An explanation has no inconsistency.
(2) A hypothesis made by abduction is explained from concepts of version spaces (Mitchell, 1977) which were generated from facts and deductive conclusions of the domain theory.
(3) An explanation should be selected that has the lowest total cost of hypothesis generation

To consider the relation between generalization level and meaning of concept representation level (such as structure information of objects and attributes of objects) , this method can also deal with semantic hypothesis selection.

After abductive explanation generation, to generalize this explanation tree, an assumptive macro knowledge is obtained by EBL.

In ICOT, a knowledge acquisition system, EPSILON (Taki, Tsubaki and Iwashita, 1987), has been developed as an interactive knowledge acquisition

system which creates an initial knowledge base based on an expert model represented by primitive operations. EPSILON has been implemented on PSI (Personal Sequential Inference Machine). The learning method by abductive explanation is developed as a learning module of EPSILON which extracts an assumptive knowledge after initial knowledge base building.

2. ABDUCTIVE EXPLANATION

When we want to explain something but there is insufficient information, we make hypotheses. These hypotheses are either uncertain knowledge or assumed knowledge. The following sections describes how to assume knowledge for explanation. In the discussion, we assume that background theory, a goal and examples are given in learning process. This background theory is not enough to explain the goal with examples.

2.1 Explanation under Incomplete Knowledge

In knowledge acquisition, it is important to understand observed data and given examples. Understanding examples means to give an explanation as to why some examples are given to realize a goal. Usually we explain a physical behavior by stating a physical law. In this case, the physical law is background knowledge and we explain the behavior as a goal with physical situations as examples. If some physical situations or a few physical law are unknown, we cannot explain the physical phenomenon. However, we can make assumptions to try to explain that phenomenon.

(Example 1) Explanation about a broken bridge

When we find a broken bridge, we wonder how the bridge broke. We didn't see it happen, so we must guess, . We assume that bad weather broke the bridge.

An explanation: Rain fell.(assumption)
The river rose.
The bridge spans the river.
The swollen river was very strong.
The river pushed against the bridge.
The river was stronger than the bridge.
The bridge broke.

Assumed information is a hypothesis. There are two types of hypotheses: one of them is in a given hypothesis set which includes uncertain knowledge (we don't

know whether the knowledge is true or false), another is generated by advanced reasoning (such as analogy, abduction or induction). The assumption of the weather in example 1 is an uncertain hypothesis. We know candidates for the weather; rain, wind, snow, fine and cloudy. We select a candidate from this finite hypothesis set. This gives us a macro knowledge: if it rains, a bridge may break. This process is like EBL without using hypotheses.

(Example 2) Explanation about a flying beetle

Tom knows that small insects can fly but he doesn't know that a beetle can fly. One day, a big flying insect flashed into Tom's head. Tom saw that the insect had a horn.Tom was surprised to see such a big insect fly. Tom assumes that any insect can fly.

An explanation: Tom knows that small insects can fly, so he induces that any insect can fly. (assumption)

Then, that flying object is a big insect.(assumption)

Then, the insect has a horn. So Tom identifies it as a flying bee

A hypothesis in example is generated by inductive inference. Tom thought that a concept of "flying insects" was generalized and a concept of "flying insects" was generated by eliminated the size attribute of insects. This example shows that hypothesis generation is important to explain examples under incomplete knowledge. A hypothesis is generated only if it is necessary.

2.2 Explanation and Hypothetical Reasoning

In explanation with inconsistencies, there may be inconsistency. Hypothetical reasoning deals with inconsistent reasoning during hypothesis selection. A formulation of a hypothetical reasoning (Poole, 1988) is as follows.

B: a background theory, H: a set of hypotheses, E: examples,
G: a goal, h: a subset of H

$h \in H$

$B \cup E \not\vdash \bot$

$B \cup E \cup h \vdash G$

$B \cup E \cup h \not\vdash \bot$

We eliminate the first formula and add the hypothesis generation formula.

$B \cup E \nvdash G$

$B \cup E \cup G \mid< h$ (Induction)

$B \cup E \cup h \vdash G$

$B \cup E \cup h \nvdash \bot$

This formula is shown in induction (Genesereth and Nilsson, 1986). We use the latter formulation. We think that this formulation is suitable not only for induction but also for abduction and analogy.

Figure 1 shows an explanation system using ATMS (de Kleer, 1986) as hypothetical reasoning.

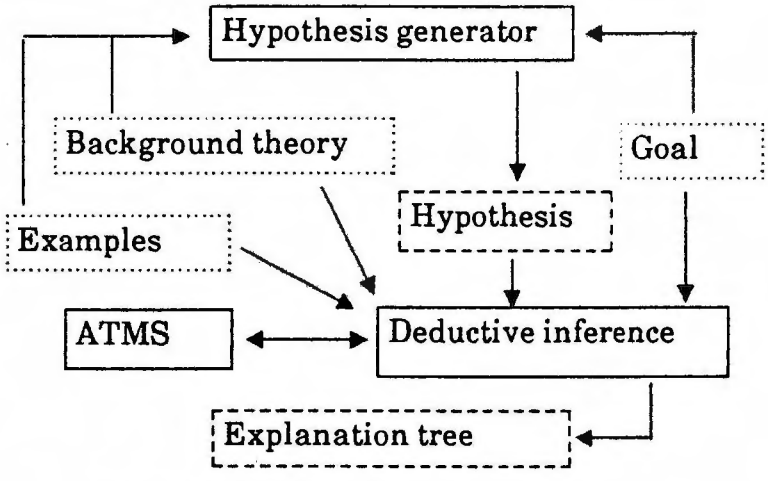


Figure 1 The explanation system

2.3 Generation of Hypothesis by Abduction

There are two kinds of candidates for hypotheses in this framework. They are fact from knowledge and rule form knowledge. We deal with the fact form knowledge as hypothesis. It is difficult to generate hypothesis deductively. This problem contains when, how and on what to generate hypotheses. When a reasoning process fails to make an explanation, this situation is a trigger to make a hypothesis. Abduction is the process that generates explanations. Abduction has the following form.

Given "if a then b"
Given "b"

Infer "a"

A prolog-like abduction form to use Horn clause is shown as follows.

Given b(X) :- a(X)
Given b(c)

Infer a(c)

Here is an example which makes an explanation by abduction.

(Example 3) Explanation about a bird, tweety
Background theory: bird(X) :- fly(X), has(X, wings), has(X, bill).
Examples: has(tweety, wings)
fly(tweety)
Goal: bird(tweety)



Figure 2 An explanation tree

Given bird(X) :- fly(X), has(X, wings), has(X, bill)
Given bird(tweety)
---------- (ABDUCTION)
Infer has(tweety, wings), fly(tweety), has(tweety, bill)

"has(tweety, wings) and fly(tweety)" are known. Therefore a result of abduction is "has(tweety, bill)". If there is the hypothesis, "has(tweety, bill)", then the explanation of example 3 is established.

3. SELECTION ABDUCTIVE EXPLANATION

If we use abduction for any explanation, we can explain anything. It is necessary to select good hypotheses in order to generate better explanations. Selection of hypotheses means selection of explanations. This section describes an evaluation as selection criteria.

3.1 Selection Hypothesis

We define the following criteria to select hypotheses.

(1) An explanation has no inconsistency.
(2) A hypothesis made by abduction is explained from concepts of version spaces which are generated from facts and deductive conclusions of the domain theory.
(3) The explanation with the lowest total cost of hypothesis generation should be selected

The second criterion selects hypotheses which are explained by induction. If there are two hypotheses in an explanation, the total cost is the sum of both hypothesis generation costs.

3.1.1 Types of Hypothesis

To begin with, types of hypothesis in the explanation system are introduced. Necessary hypothesis is dependent on a sequence of explanation. In example 3, if a goal concept is bird(X), there is a difference between rule 1 "bird(X) :- fly(X), has(X, wings), has(X, bill)" and rule 2 "bird(X) :- has(X, bill), fly(X), has(X, wings)".

In an inference using the rule 1, when "has(X, bill)" is checked, the value "X" has been unified to "tweety" which was fixed at checking "fly(X)". Therefore, a result of abduction in this case is "has(tweety, bill)". On the other hand, when "has(X, bill)" is checked in the case of the rule 2, the value "X" is not fixed. The result of abduction is "has(X, bill)". After reasoning, both explanations are the same. In the second case, as a result, "has(X, bill)" is "has(tweety, bill)". This

shows that necessary hypothesis is "has(tweety, bill)" in both cases. We call this result of abduction "has(tweety, bill)" a basic hypothesis and the result "has(X, bill)" a temporal hypothesis.

If there are a basic hypothesis and a temporal hypothesis from an item by abduction in an explanation, the temporal hypothesis is more general than the basic hypothesis and the basic hypothesis is the most special hypothesis in the explanation.

3.1.2 Version Space and Hypothesis

Knowledge in the background theory can explain only concepts that are more specific than itself. Generalized knowledge is necessary to explain a hypothesis. Sets of generalized knowledge are represented by version spaces made from the background theory.

Normally, the induction function decides an upper and lower boundary of a version space using positive and negative examples of a concept.

If the background theory doesn't include negative examples, version spaces of concepts in the background theory have no upper boundaries. Therefore, the most generalized form of an item has its own predicate name and arguments that are represented by valuables. This means that the predicate name is not generalized. The lower boundaries of those version spaces are defined by facts that are deduced from the background theory.

A concept in a version space which explains a basic hypothesis is called "a support concept for the basic hypothesis". The most specific concepts of support concepts is called "the least generalized support concept". The least generalized support concept is one of concepts made from the basic hypothesis and facts in the background theory by least generalization.

(Example 4) The least generalized support concept, a version space and an explanation tree

In example 4, "has(fish, fins)" is the lower boundary of the version space, "has(X,Y)" is the upper boundary of it. "has(X, fins)" and "has(fish, Y)" are concepts in the version space. If "has(dolphin, fins)" is a basic hypothesis, "has(X,Y)" and "has(X, fins)" are support concepts for the hypothesis because these unify to it. A least generalized support concept is "has(X, fins)".

If "has(beetle, horn)" is a basic hypothesis, a least generalized support concept is

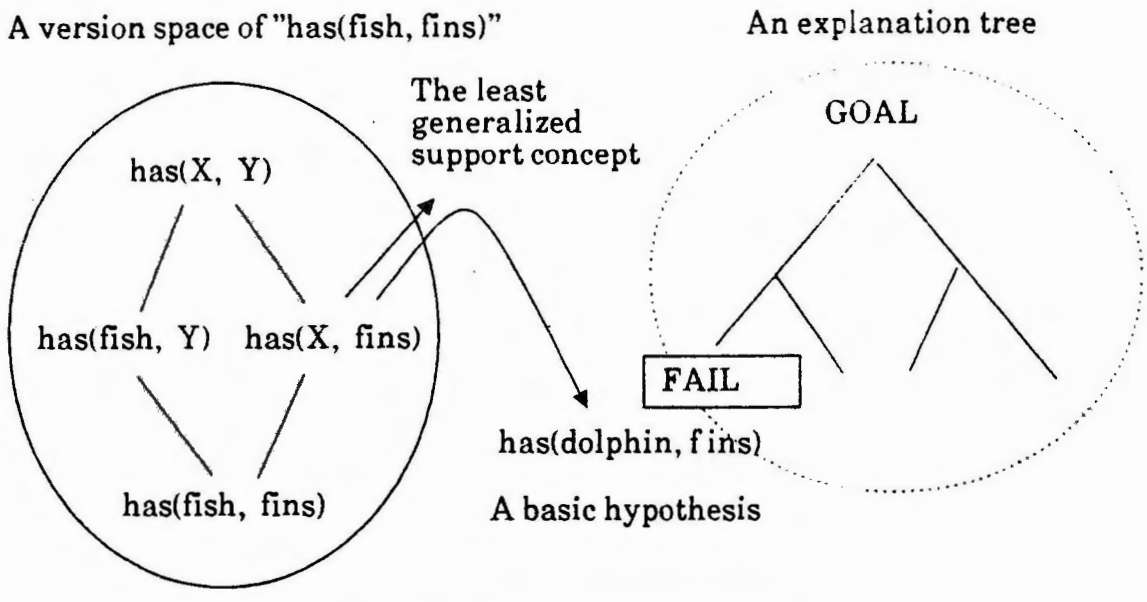


Figure 3 The least generalized support concept

"has(X, Y)". If "eat(dolphin, fish)" is a basic hypothesis, there is no support concept for this hypothesis in this version space.

3.1.3 Hypothesis Generation Cost

This section describes operations for predicate modification. Each operation has the same cost and is identified as 1 point. Three operations are defined here. The first operation, argument generalization operation (G-op), exchanges a fixed argument to a variable. The second operation, argument disconnect operation (D-op), eliminates a dependency between two variables. The third operation, argument unification operation (U-op), unifies a instance and a variable.

(Example 5) Operations

G-op: has(dolphin, fins) -> has(X, fins) or has(dolphin, Y)

D-op: has(X,X) -> has(X,Y)

U-op: has(X, fins) -> has(fish, fins)

In example 4, total cost of generation "has(dolphin, fins)" from "has(fish, fins)" is 2 points. This generation process consists of "G-op: has(fish, fins) -> has(X, fins)" and "U-op: has(X, fins) -> has(dolphin, fins)".

4. ASSUMPTIVE MACRO KNOWLEDGE GENERATION

In this section, this explanation function is integrated into EBL. We can obtain assumptive macro knowledge by abductive explanation-based learning.

4.1 Explanation-Based Learning and Incomplete Domain Theory

EBL is a learning about efficiency of knowledge. In EBL, complete domain theory, goal concept, learning examples and operationality criteria must be prepared. EBL explains a goal using examples and domain theory. An explanation tree is generalized by operationality criteria. Before learning, it is hard to prepare a complete domain theory. If a framework of EBL under incomplete domain theory were developed, it could learn new knowledge and efficiency of knowledge usage. There is some incompleteness in the domain theory as follows.

(1) Completeness: There is some lack of knowledge, so some examples cannot be explained.

(2) Soundness: There is some wrong knowledge, so wrong examples are explained.

(3) Consistency: The domain theory contains inconsistency, so inconsistency is detected in an explanation.

(4) Over-generalization: Knowledge is too general, so it explains negative examples.

(5) Over-specialization: Knowledge is too special to explain positive examples.

We aim to develop this framework using abduction and induction. We aim to develop an explanation system which deals with incomplete knowledge (1) and (5). It makes assumptive explanations and generalizes explanations. As a result, it generates assumptive macro knowledge which contains hypothesis as a new knowledge and usage of the hypothesis.

4.2 Assumptive Macro Knowledge

A basic hypothesis is very special for learned knowledge. A least generalized support concept is a support hypothesis for a basic hypothesis. Therefore, we use the least generalized support concept as a hypothesis to learn new knowledge. If the support concept is denied, the basic hypothesis is also denied. The support concept may explain other hypotheses different from the basic hypothesis.

However, examples and an explanation are special cases. The learning system must learn more general inconsistent knowledge. We define assumptive macro knowledge shown in a following form.

Knowledge generated by EBL:

goal(X1, ..., Xn) :- p1(Xi, .., Xj, ..), p2(...), ..., pm(...). --(1)

Hypothesis: h1(Xk, dolphin).

Assumptive macro knowledge (represented in default rule form):

p1(...), p2(...), ..., pm(...) M: h1(Xk, dolphin)

---

goal(X1, ..., Xn)

Unless "h1(Xk, dolphin)" is not denied, a horn clause (1) is available.

5. SEMANTIC HYPOTHESIS SELECTION

The assumptive macro knowledge is generated syntactically. This section shows how we can express semantics about generalization and specialization.

5.1 Generalization Level of Hypothesis

The least generalized support concept is generated from the background theory and examples by generalization. However, there are some limits for generalization. These limits decide the upper boundaries of version spaces made from known knowledge such as negative examples. These constraints are dependent on application fields. For example, structured mapping theory (Falkenhainer, Forbus and Gentner, 1987) controls analogical mapping between two events. It maps structure information but not attributes of objects in these events. In order to control generalization levels of special predicates, this system allows the user to limit version spaces of predicts which are defined in a set of special names or have a fixed number of arguments.

5.2 Operationality

In order to generate operational macro knowledge, EBL generalizes an explanation tree about its structure. In our learning method, there are two relations between a hypothesis and an operationality criterion.

(1) An operationality criterion is more special than a hypothesis.

In this case, the hypothesis exists in macro knowledge. If this macro is used in an inference, this hypothesis must be explained by deduction. This type of macro knowledge is normal knowledge but not default knowledge.

(2) An operationality criterion is more general than a hypothesis.

In this case, the hypothesis doesn't exist in macro knowledge. It is eliminated from the explanation tree by an operatinality criterion. Therefore, this macro is independent of knowledge whose generalization level represents the hypothesis. The hypothesis is no longer necessary to explain the macro knowledge. However, if the hypothesis is denied, the macro knowledge loses its generation cause/reason. This type of knowledge is represented in assumptive macro knowledge form.

6. EXAMPLE OF DESIGN KNOWLEDGE ACQUISITION

Knowledge acquisition is more difficult in synthetic problems than in analytic problems. Design tasks are synthetic problems. A designer uses a great deal of knowledge for design tasks; it is difficult to remember everything. We observed that human designers learn basic design by solving simple examples. Then they use the basic knowledge for more complex design objects and, in so doing, learn new knowledge. We thought that the way human designers learn suited our knowledge acquisition method. We tried to apply our method for reduction gear systems design.

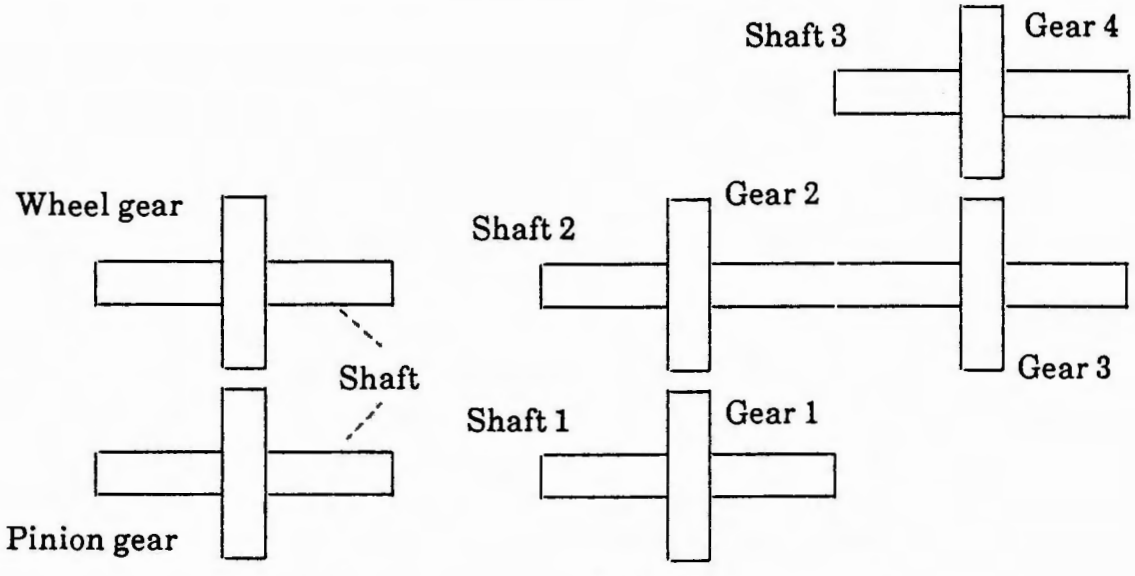
6.1 Reduction Gear System Design

We have been developing a constraint knowledge compiler, MECHANICOT (Terasaki, Nagai, Yokoyama, Inoue, Horiuchi and Taki, 1988), which analyzes constraints and makes a data-flow network. Output of the MECHANICOT is an expert system for reduction gear system design. Currently, this system has design knowledge for a reduction structure which includes two axes, four gears and a double-reduction gear mechanism. We try to acquire knowledge from which triple-reduction gear systems can be designed. Our knowledge acquisition strategy is shown as follows,

(1) Understanding the structure of an example complex object by using knowledge of the structure of simple objects

(2) Designing an example complex object by using design knowledge of simple objects

These steps are based on a multiple hierarchical model for design objects (Katai, Kawakami, Sawaragi, Iwai and Shibata, 1988). This model consists of four spaces (purpose set, function set, structure attribute set and real value/object set) and three knowledge spaces (plan type knowledge, physical rule knowledge and realization knowledge). A purpose set presents design requirements. The plan type knowledge makes a function set from the purpose set. The physical rule knowledge makes a structure attribute set from the function set. The realization knowledge makes a real value/object set from the structure attribute set. Our example shows explanation of the structure attribute set and the real value/object set using abductive and inductive explanation. Figure 4 shows an example of reduction gear systems. Structure and design knowledge of reduction gear 1 (Figure 4 (a)) is given. The system tries to acquire structure and design knowledge of reduction gear 2 (Figure 4 (b)). Reduction gear 1 has two axes and two gears but no speed change mechanism. Reduction gear 2 has three axes and four gears but no speed change mechanism.



(a) Reduction gear 1 (b) Reduction gear 2

Figure 4 Reduction gear systems

6.2 Explanation for Structure of Reduction System

The system tries to understand reduction gear 2 as a kind of reduction gear. The system has structure knowledge of reduction gear 1 as domain theory.

[Domain theory]

```
shaft(input, X, Y) :- on(X, Y), gear__type(pinion, Y).
shaft(output, X, Y) :- on(X, Y), gear__type(wheel, Y).
reduction__gear1(A1, A2, G1, G2) :- shaft(input, A1, G1),
   shaft(output, A2, G2), gear__pair(G1, G2).
```

[Goal concept]

```
reduction__gear2(A1, A2, A3, G1, G2, G3, G4) :-
   reduction__gear1(A1, A2, G1, G2), reduction__gear1(A2, A3, G3, G4).
```

[Example]

```
gear__type(pinion, gear1). gear__type(middle, gear2).
gear__type(middle, gear3). gear__type(wheel, gear4).
on(gear1, shaft1). on(gear2, shaft2). on(gear3, shaft2).
on(gear4, shaft3). gear__pair(gear1, gear2). gear__pair(gear3, gear4).
```

The system fails to explain these goal concepts. The basic hypotheses are "gear__type(wheel, gear2)" and "gear__type(pinion, gear3)". The system makes basic supports concepts; "gear__type(Any1, gear2)" and "gear__type(Any2, gear3)". This explanation shows that it is necessary to generalize the gear types of reduction gear systems in order to understand a new example.

6.3 Explanation for Design Process of Reduction System

Reduction gear 2 consists of two reduction gear system which are same type as reduction gear 1. Figure 5 shows this structure which includes A part and B part as reduction gear 1.

[Design knowledge of reduction gear 1]

Goal: design__g1(reduction__ratio, a set of pinion__gear, Wheel__gear)

Knowledge:

```
design__g1(RR,[],[]).
```

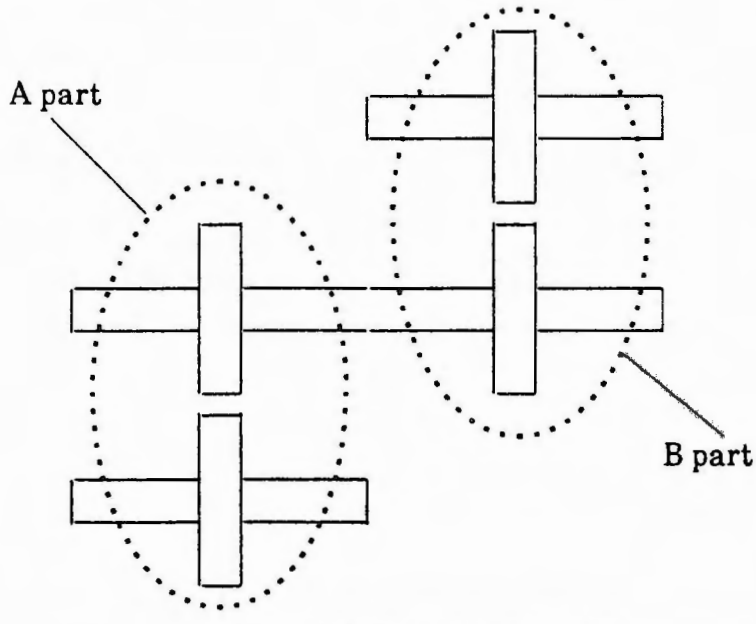


Figure 5 Reduction gear 2

```
design__g1(RR, [Pinion__gear|Rest], Wheel__gear):-
   gear__select(RR, Pinion__gear, Wheel__gear).
design__g1(RR, [Pinion__gear|Rest], Wheel__gear):-
   design__g1(RR, Rest, Wheel__gear).
gear__select(RR, Pinion__gear, Wheel__gear):-
   gear__teeth(Pinion__gear, Th1), Th2 = RR/Th1,
   gear__teeth(Wheel__gear, Th2).

gear__teeth(g11, 100).
gear__teeth(g12, 120).
   :
gear__teeth(g21, 26).
gear__teeth(g22, 14).
   :
```

The system tries to design reduction gear 2 by using design knowledge of reduction gear 1. Reduction gear 2 consists of the A part and the B part shown in Figure 5. Therefore, the following knowledge can be derived from the structure

explanation tree. This system uses a heuristic which makes some relations between parts, when it separates a structure into parts.

```
Goal: design__g2(RR, a set of g1, Gear2, Gear3, Gear4)
design__g2(RR, [], [], [], []).
design__g2(RR, [G1|Rest], G2, G3, G4) :-
        gear__select(RR2, G1, G2), gear__select(RR3, G3, G4),
        relation between the A part and the B part.
design__g2(RR, [G1|Rest], G2, G3, G4) :- design__g2(RR, Rest, G2, G3, G4).
```

In this case, the relation between the A part and the B part is shown by the formula "RR = RR2 * RR3". Currently, the system cannot make this formula and asks the designer for it. Even if this formula is given, the system fails to explain the goal of "design__g2" because RR2 and RR3 are not unified, so variable Th2 in "gear__select" cannot be calculated. RR is output, and RR2 and RR3 are input in this formula, so it is a special form. Therefore, the system makes a generalized formula "f(RR, RR2, RR3)" which has no constraint of variables regarding input and output. The new formula calculates RR2 and RR3, so a sub goal "gear__select" can be succeeded in the explanation process. The basic hypothesis is "a relation between the A part and the B part". The generalized hypothesis is a function "f(RR, RR2, RR3). The system can only show this function, but more examples must be given to decide this function. As a result, the system asks the designer for more information to decide this function by showing the features of this function (input or output variables). In the real world of design reduction systems, this function "f(RR, RR2, RR3)" selects a pair ratio by using a geometrical series table.

7. RELATED WORKS

Our research is a kind of EBL under incomplete knowledge (Doyle, 1986; Rajamoney, 1988; Uchihashi and Taki, 1989). A characteristic of this learning is to explain by abduction and induction. The research of abductive explanations in natural-language interpretation in SRI (Stickel, 1988) also discusses abductive explanation and assumption cost. Its explanation is based on abduction. We use abduction to make a basic hypothesis and decide least generalized support concept using a version space. Therefore, our explanation is based on induction. The integration of EBL, and ATMS or hypothetical reasoning (Atsumi, 1988) is

developed for making consistent explanations. Our system also uses ATMS to check explanation consistency. A default rule maintenance system (Araragi and Ishida, 1989) uses a trigger when an inconsistent explanation is found. It modifies a default representation of a default rule. Our system doesn't maintain default rules, but eliminates inconsistent explanations and macro knowledge. The learning method to generalize explanation trees with the integration method of similarity-based learning and operationality criteria is developed (Yamamura and Kobayashi, 1987). Our system uses induction to generate explanations but not to generate a generalized explanation tree from some explanation trees. A model inference system (Shapiro, 1981) uses a refinement tree to specialize predicates with six strategies. The current version of our system uses only three operations to generalize and specialize knowledge. Its next version supports the following operations.

List generalization operation 1: has(X,[Y|Z]) -> has(X,Y)
List generalization operation 2: has([X|Z],Y) -> has(X,Y)
Implication eliminate operation: has(X,Y) :- has(X,Y) -> has(X,Y)

This system deals with fact form hypotheses. The inverting resolution (Muggleton and Buntine, 1988) method might be useful to generate rule form hypotheses. Our method generalizes the background theory and examples to make version spaces and specializes a least generalized support concept to explain a basic hypothesis. This reasoning is a kind of analogy.

8. SUMMARY

This paper discussed a learning method about how hypothetical knowledge is extracted. This method uses an advanced reasoning which generates abductive explanations and selects hypotheses according to checking costs of hypothesis generation. A basic hypothesis is made by abduction when an explanation is not generated. A least generalized support concept is also derived from the basic hypothesis and version spaces which are made by a background theory and examples. This support concept is explained by induction of known knowledge. A assumptive macro knowledge is made from the explanation with this support concept. This knowledge is available unless the support knowledge is not denied. A current version of this method deals with facts as hypotheses. It is necessary for us to develop rule generation methods to explain examples and an integrated

knowledge acquisition system (EPSILON) which consists of an interview system and a learning system.

9. ACKNOWLEDGEMENT

We wish to express our thanks to the member of the Knowledge base System Architecture - Knowledge Acquisition and Design - Sub Working Group , to its head, Professor Riichirou Mizoguchi and to Professor Norihiro Abe. We would like to thank the members of the Learning and Non-monotonic Reasoning System Research Group at ICOT and Dr. Koichi Furukawa. We give thanks to the members of 5th laboratory at ICOT, Mr. Katsumi Inoue, Mr. Satoshi Terasaki and Mr. Eiichi Horiuchi, the invited researcher, Mr. Marc Linster (GMD), and the chief of 5th laboratory at ICOT, Mr. Kenji Ikoma. Finally, we give our thanks to Dr. Kazuhiro Fuchi, Director of ICOT Research Center, who provided us with the opportunity to conduct this research in the Fifth Generation Computer Systems Project.

References

Araragi, T. and Ishida, T. (1989) Explanation-based Refinement of Knowledge Base, JSAI SIG-KBS-8805 (in Japanese).

Atsumi, M. (1988) A View of EBL in a Hypothesis Reasoning System, Japan Software Science Society, Proc. 5th Conference (in Japanese)

Charniak, E. and McDermott, D. (1986) Introduction to Artificial Intelligence, Addison Wesley

de Kleer, J. (1986) An assumption-based TMS, Artificial Intelligence 28, pp.127-162

Doyle, R.J. (1986) Constructing and Refining Causal Explanations from an Inconsistent Domain Theory, AAAI-86, pp.538-544

Falkenhainer, B., Forbus, K.D. and Gentner, D. (1987) The Structured-mapping Engine: Algorithm and Examples, Technical Report, UIUCDCS-R-87-1361, Dep. of Computer Science, University of Illinois

Genesereth, M. and Nilsson, N. (1986) Logical Foundations of Artificial Intelligence, Los Altos, CA: Morgan-Kaufmann

Inoue, K. (1989) Towards a Logical System for Hypothetical Reasoning, 7th ICOT symposium proceeding, June (in Japanese)

Katai, O., Kawakami, H., Sawaragi, T., Iwai, S. and Shibata, K. (1989) Design Knowledge Acquisition by Examples, Proc. of 28th SICE Annual

Conference, JSS 3-12 (in Japanese)
Laird, J., Rosenbloom, P. and Newell, A. (1986) Universal Subgoaling and Chunking, Kluwer Academic Publishers
Mitchell, T. (1977) Version spaces: A candidate elimination approach to rule learning, Proc. 5th IJICAI,pp.305-310
Mitchell, T., Mahadevan, S. and Steinberg, L. (1985) LEAP: A Learning Apprentice for VLSI Design, Proc. 9th IJICAI, Los Angeles
Mitchell, T., Keller, R., and Keder-Cabelli, S. (1986) Explanation-Based Generalization: A Unifying View, Machine Learning 1
Muggleton, S. and Buntine, W. (1988) Machine Invention of First Order Predicates by Inverting Resolution, Proc. 5th Machine Learning, pp339-352
Poole, D. (1988) A Logical Framework for Default Reasoning, Artificial Intelligence 36, pp.27-47
Rajamoney, S.A. (1988) Experimentation-Based Theory Revision, AAAI Spring Symposium Series: Explanation-Based Learning, pp. 7-16
Shapiro, E.Y. (1981) Inductive inference of theories from facts, Technical Report 192, Yale University Computer Science Dept
Stickel, M.E. (1988) A Prolog-like Inference System for Computing Minimum-Cost Abductive Explanations in Natural-Language Interpretation, SRI International Tech. Note 451, Sept.
Suwa, M. and Motoda, H. (1989) Understanding Metaphor by Frustration-Based Learning, Proc. SICE 9th Knowledge Engineering Symposium (in Japanese)
Taki, H., Tsubaki, K. and Iwashita, Y. (1987) Expert Model for Knowledge Acquisition, Proc. IEEE 3rd Annual Expert Systems in Government Conference, Washington D.C.
Terasaki, S., Nagai, Y., Yokoyama, T., Inoue, K., Horiuchi, E. and Taki, H. (1988) Mechanical Design System Building Tool: MECHANICOT, JSAI Proc. of SIG-KBS-8803-3 (in Japanese)
Uchihashi, T. and Taki, H. (1989) An Interactive EBG System, ICOT Technical Report No.471 (in Japanese)
Yamamura, M. and Kobayashi, S.(1987) An Integration of SBL and EBL based on Operationality Criteria, SICE Proc. 1st Knowledge Engineering Meeting (in Japanese)

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

New Learning Algorithm for Rule Extraction by Neural Network and Its Application

J. TANI, T. HIROBE, K. NIIDA and I. KOSHIJIMA
CHIYODA CORPORATION
12-1, Tsurumichuo 2-chome, Tsurumi-ku, Yokohama, Japan
H. MURAKAMI
NIPPON FELT CO., LTD.
88, Haramarumo, Konosushi, Saitama 365, Japan

ABSTRACT

Major questions in neural network research are 1) how to extract explicit knowledge from the connections obtained, and 2) how to determine the size of the network needed to learn a particular target. To address these questions, the authors analyze the internal representation features of the PDP model in regard to the complexity of the network's connections. Based on this analysis, a new learning algorithm to control the connection distribution as well as the learning error is developed. Tests by simple problems and application to a design problem show that the proposed method can derive quite explicit representation with a smaller number of connections.

1. INTRODUCTION

Recently, studies of neural networks have attracted much attention in the AI area of computer science. More specifically, the Perceptrons (Minsky, 1969) type network called the PDP model (Rumerhart, 1986) is considered to have enormous potential in the field of artificial learning. The model, however, contains numerous unsolved problems. Foremost among them are the following two. 1) The network cannot show what it has learned in reasonably explicit forms. The inside of the network is quite "muddy" with the association of complex connections. 2) We cannot tell what will be the optimal complexity of the network (i.e., number of connections) and still assure the accuracy of the network's output. It can be said intuitively, however, that representation of complex features requires a network with a large number of connections while the representation of simple features requires a network with a small number of connections. An

investigation of the network's representation nature in respect to the network's level of complexity seems to be the key to approaching both problems.

Any representation scheme addresses a general subject; how to achieve the highest level of refinement in its representation with the smallest number of descriptions (Minsky, 1981) (Woods, 1986). It is worth considering the schema, a representation methodology in AI domain as an example. The schema minimizes redundancy in its representation by gathering common features and sharing them in one memory location, while separating distinct features and attaching them to different memory locations. The representation of the schema is quite efficient in that what is common and what is distinct among objects are easily retrieved (Norman, 1976). Efficiency is, in other words, measured by cost of retrieval. For the PDP model, where a connection is regarded as a description element, if a network needs to activate a greater number of units, propagating through a greater number of redundant connections for retrieval of certain information, the representation is said to be inefficient. It can be considered that "explicit representation" is a representation with minimum redundancy. As the redundancy is minimized, each description element is forced to have its own role orthogonal to others, functioning as an indispensable component in the whole representation scheme. Explicit representation does not always mean a representation consisting of a small number of descriptions. To represent up to a quite detailed level, more descriptions at that level are necessary. These descriptions are not redundant but are necessary to achieve the specified refinement. Each level of refinement determines the least number of indispensable descriptions. As in the case for the PDP model, more refinement in representation requires more connections describing details, while abstraction leads to fewer connections describing only general features.

Assuming that the complexity of the network could be controlled as well as its learning error by a certain learning algorithm, the relation shown in Fig.1 can be hypothesized. In the network of the most complex connections, point (1), the accuracy obtained (expressed by the mean-square error) in the training of the network can be quite good, but the input-output relations tend to be implicit with redundant connections. When the network is trained to the more simplified level of the point (2), a most refined representation with the minimum redundancy in connections is achieved. If the network is simplified further to point (3), the abstraction process of the the network results in a "skeleton shape" network. Such a network represents only the primal features, diminishing the connections describing details. At this point, an abstracted general relationship of input-output can be seen explicitly in

the connections obtained at the cost of a slight increase in learning error.

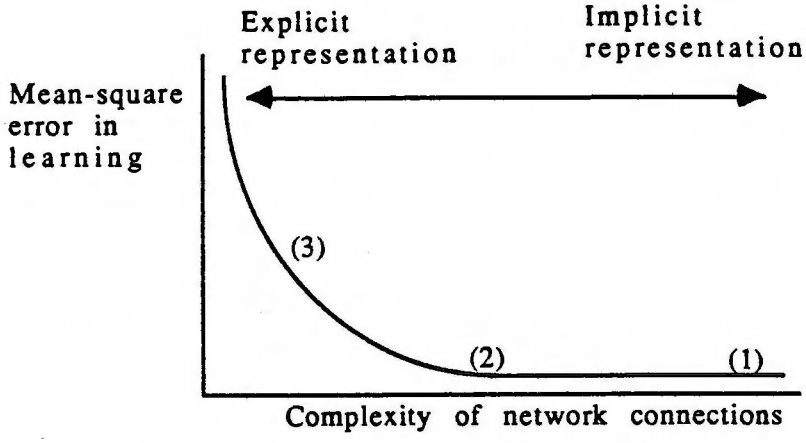


Fig.1 Relations among learning error, explicitness and complexity

This study proposes an approach to controlling network complexity by adding to the conventional back-propagation algorithm a new algorithm: the connection distribution control. The study shows that this new learning algorithm can successfully derive explicit internal representation of networks with minimum complexity.

2. NEW LEARNING METHOD

2.1 The learning algorithm

Learning in the PDP model can be considered as a process of searching the optimal network connections to minimize learning error. In the objective function of this new learning algorithm an additional term, named the distribution factor, that expresses the complexity of connections in the network is added to the mean-square error term. This learning algorithm minimizes both the learning error and the distribution of connections. The objective function is described as follows.

Objective function:

$$E = Ee + Ed$$

$$= \sum_i (t_i - a_i)^2 + c \sum_j \left| \prod_k w_{jk} \right|$$

Ee: summation of mean-square error

Ed: distribution factor

t_i: target value of *i*th output unit

a_i: activation value of *i*th output unit

w_{jk}: value of connection weight from unit *k* to unit *j*

c: distribution coefficient

Here, the local distribution factor is calculated for each unit as the absolute product over all the weights of inflow connections to the unit. The distribution factor of the whole network is the summation of the local distribution factors over all units. When the derivative of the objective function is taken at each connection weight, the derivative of the first term is the same as those shown in the generalized delta rule (Rumerhart, 1986) while the derivative of the second term is expressed newly as follows.

$$\partial E_d / \partial w_{jl} = \left| \prod_k w_{jk} \right| / w_{jl}$$

The update policy of each weight to minimize the objective functions is described as

$$\Delta w_{jl} = -e \left(\partial E_e / \partial w_{jl} + c \, \partial E_d / \partial w_{jl} \right)$$

Each unit tries to minimize its distribution factor by first decreasing the weight of any weaker connections to the unit, while trying to maintain the mean-square error as the minimum. When the value of the objective function is reduced enough to minimize both the learning error and the distribution factor, only the minimum number of connections necessary to represent a reasonable model of the target remain. In this method, the distribution factor of the network can be controlled to any level just by changing the distribution coefficient in the above equation.

A numerical technique is, however, necessary to implement this algorithm. By this algorithm, when one of the connection weights at a unit becomes 0, the local distribution factor of the unit also becomes 0, and the driving force to decrease other weights at the unit is lost. To avoid this, each connection weight is limited to be not smaller than a small predetermined value such as 0.1. When the connection weight reaches this value, the connection is considered to have disappeared.

2.2 The Learning Procedure

By using the proposed algorithm, the learning of the network proceeds from the abstract level to the refined level along the curve of Fig.1. First, the network is trained with a larger distribution coefficient so as to become a "skeleton shape" network. Once the abstract representation is obtained, training continues with the setting of smaller distribution coefficient. In this way, the representation of the network is refined further from the abstract one already obtained. Training is continued until the most refined network with minimum redundancy is achieved.

3. TESTS

3.1 Boolean Problem with Noise

A test with a boolean problem was developed to evaluate the proposed approach. The simple boolean expression shown below was chosen for a target model to be learned.

A & B => a & b
B & C => c & d

To test the robustness of the algorithm against a noisy environment, a certain extent of noise was added to the training data set as shown in Table 1.

First, the conventional back-propagation method is used to train the network. The resulting connections of the network are shown in Fig.2. Here, the connections in the network are so complex that no input-output relations can be seen explicitly. The mean-square error of the network is obtained as 0.1. Next, the proposed algorithm is applied and the resulting skeleton network is shown in Fig.3. Only a few connections remain, the others being diminished by the abstraction process. The mean-square error of the network increases slightly to 0.14. These connections show two rules explicitly. The input units of "A" and "B" are connected to the middle unit in the hidden layer which is connected to the output unit of "a" and "b". This set of connections expresses the first rule of the model: "A & B => a & b". Similar

INPUT			OUTPUT			
A	B	C	a	b	c	d
0.11	0.06	0.15	0.16	0.07	0.03	0.08
0.07	0.02	0.91	0.11	0.15	0.11	0.15
0.09	0.93	0.12	0.13	0.02	0.13	0.06
0.04	0.89	0.95	0.10	0.07	0.91	0.96
0.92	0.23	0.07	0.12	0.04	0.06	0.12
0.91	0.07	0.96	0.04	0.16	0.11	0.07
0.92	0.93	0.12	0.97	0.94	0.04	0.08
0.89	0.96	0.92	0.85	0.97	0.95	0.86
0.11	0.06	0.15	0.16	0.07	0.15	0.02
0.12	0.09	0.97	0.18	0.05	0.13	0.07
0.13	0.83	0.04	0.13	0.07	0.06	0.17
0.04	0.99	0.94	0.12	0.06	0.97	0.96
0.96	0.03	0.09	0.04	0.09	0.06	0.12
0.97	0.23	0.86	0.07	0.18	0.14	0.19
0.98	0.88	0.12	0.92	0.98	0.07	0.12
0.92	0.90	0.89	0.89	0.91	0.97	0.92

Table 1 Learning data for boolean problem

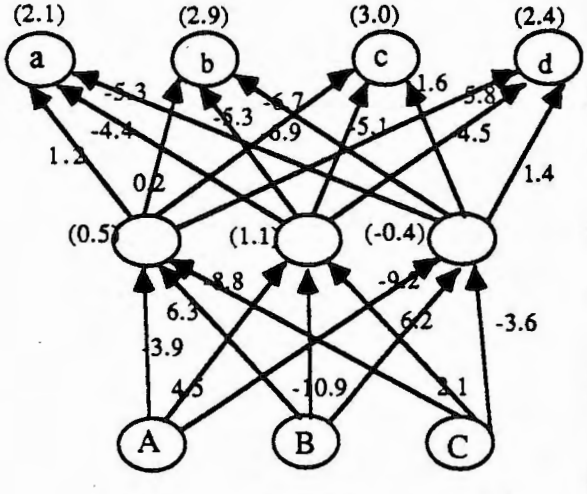


Fig.2 Connections obtained by conventional back-propagation algorithm

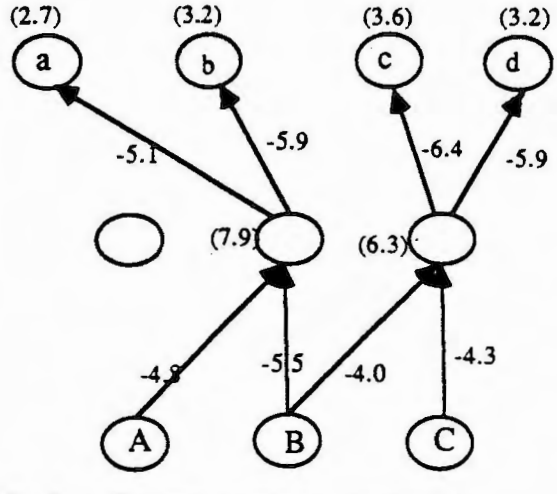


Fig.3 Skeleton network obtained by the proposed algorithm

Note : The value shown with each connection is the weight of the connection. The value inside ( ) is the bias of the unit.

connections are seen among the units of "B", "C", the right-hand unit in the hidden layer, and "c" and "d" in the output layers, those express the second rule: "B & C => c & d". The important observation here is that two sets of obtained connections represent each rule locally. The reduction of network complexity derives local representation in the network.

One difficulty that arises here, however, is that the proposed algorithm faces the local minimum problem sometime. The proposed algorithm generates two other styles of skeleton network such as shown in Fig.4.

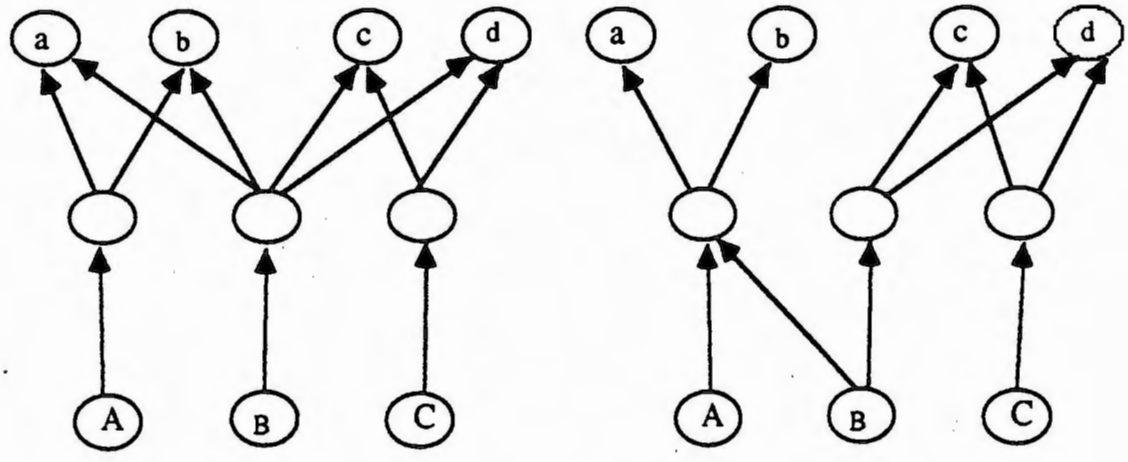


Fig.4 Other obtained skeleton networks which are trapped in local minimum

These skeleton networks contain some more connections than the network shown in Fig.3 while resulting almost the same mean-square error. These have some redundancy in their representations, being trapped in the local minimum of the distribution factor. Achieving the truly optimized connections along the line between (2) and (3) of Fig.1 might cost larger calculations in case of learning complex targets. It is, however, considered that achieving the sub-optimal connections are enough and they are still explicit and less redundant in their representation.

3.2 Schema Problem

The second example shows learning of the simple schemata: "dogs" and "cats". Suppose that there are four animals. We can observe some features of these animals: their food preferences for eat and their cries. Our observations might be summarized as in Table 2. The features of each animal are expressed by analog values between 0 and 1 depending

Animal Instances	Features of Animals			
	prefer bone	prefer fish	cry bow-wow	cry meow-meow
instance 1	1.0	0.0	0.75	0.25
instance 2	0.75	0.25	1.0	0.0
instance 3	0.0	1.0	0.25	0.75
instance 4	0.25	0.75	0.0	1.0

Table 2 Features of animals

on how relevant the feature is to each animal-instance. For example, Animal-instance 1 prefers bone quite strongly at a level of 1.0, while instance 2 does so a little less at 0.75. We can see that the first two instances can be categorized into "dogs" and the last two into "cats" from these features. Now we let the neural network learn this. The network for training consists of four input units representing animal instances, four output units representing features, and four hidden units. By using the proposed learning method, we obtained two networks of different complexities.

Training of the network to reduce its complexity to skeleton level, shown in Fig.5, derives the connections with a learning error of 0.17.

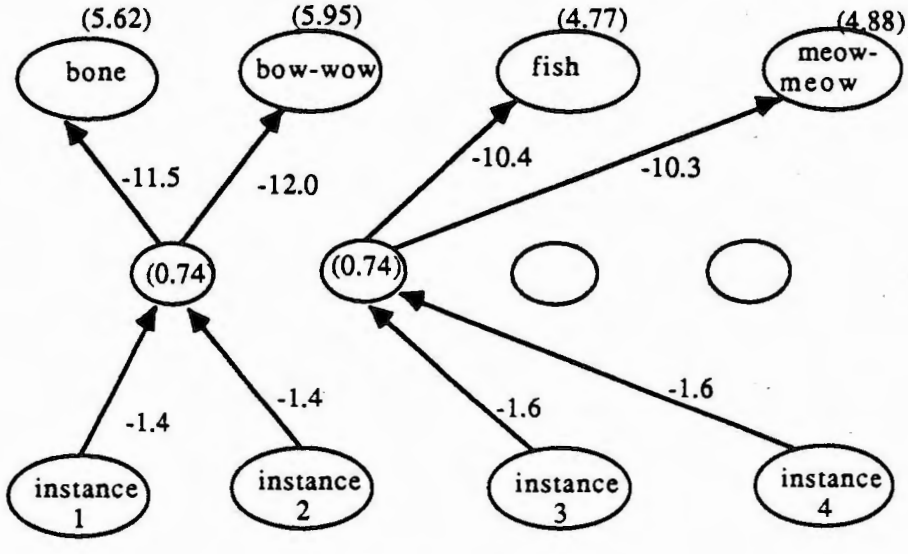


Fig.5 Skeleton network
(Mean-square error is 0.17)

The hidden unit 1 is connected from input units of instance 1 and instance 2 to the output units of "bone" and "bow-wow". These connections can be said to represent the schema of "dogs", sharing "dogs" features in this local connection. In the same way, the right side connections including hidden unit 2 can be said to represent the schema of "cats". In this network, the representation is quite abstract expressing only the difference between the two major categories "cats" and "dogs" and ignoring the detailed difference among instances in the same category.

Next, allowing the network's complexity to increase slightly generates the most refined network, with a learning error of 0.003, as shown in Fig.6. An additional hidden unit appears at the right, connecting from/to all instances and features. This set of connections compensates minor differences missing in the previous network. The representation of the network obtained is quite efficient in that what is common and what is different are shown quite explicitly.

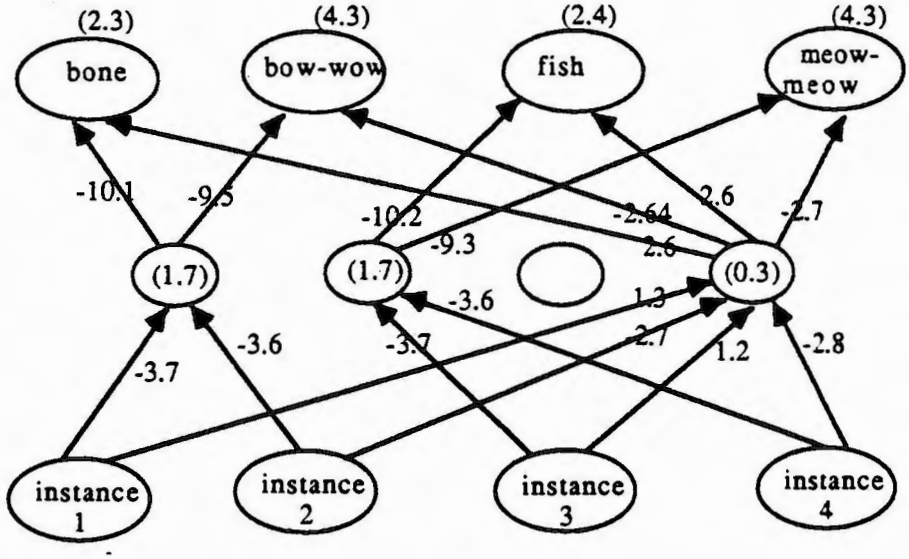


Fig.6 Most refined connections with minimum redundancy (The mean-square error is 0.003)

4. APPLICATION

The proposed method is applied to a knowledge acquisition task in felts design. In paper manufacturing, felts are indispensable components. They support the water slurry of pulp from underneath, absorbing water and assisting the formation of the multilayer fabric (see Fig. 7). The felt plays multiple roles in this process, requiring multidimensional qualities. The design of felts to achieve the objective

qualities is conducted by design experts. However, the many of design procedures depend on each expert's experience, not integrated as the general rules. The application task described below is to extract general knowledge which explains the causality between design conditions and the product's qualities from the past accumulated design data.

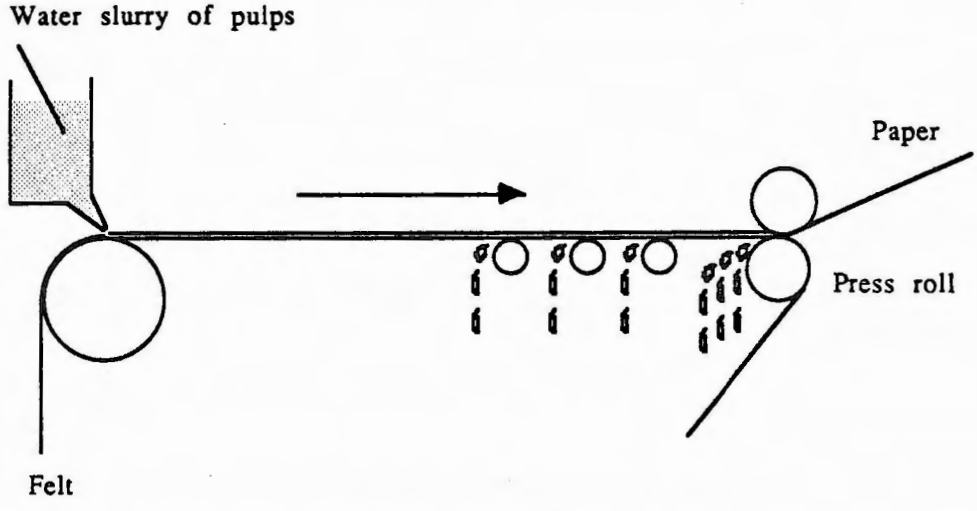


Fig. 7 A part of paper manufacturing process

The network used for the training consists of 12 input units, eight hidden units and two output units. Input units are assigned for design conditions such as selection of needles, chemical treatments and setting of mechanical parameters. For output units, the qualities of the felts as measured by drainability and surface smoothness are assigned. In the actual input, the selection of needles and chemical treatments are made on a 0 or 1 base (i.e., if the one is selected, the input is 1; if not, 0). For mechanical parameters, each is mapped to an analog value between 0 and 1. The output units of product quality are mapped to analog values between 0 and 1, with 1 as good quality and 0 as not.

Firstly, the conventional back-propagation method is applied to the training of the network. A network with the mean-square error of 0.376 is obtained.

The proposed algorithm is applied to reduce the distribution factor of the network. A "skeleton shape" network consisting of a small number of connections with a mean-square error of 0.573 is obtained. Shown in Fig.8, it consists of three local connections with three hidden units, explaining the input-output relations explicitly. The second hidden unit is connected positively to the drainability unit and the surface smoothness unit. The connections represent a good factor for

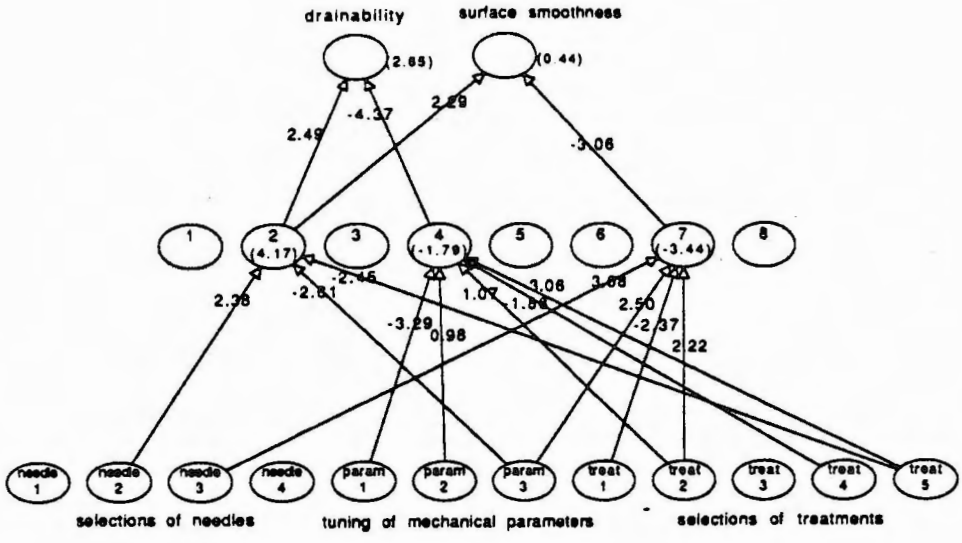


Fig.8 Skeleton network for the application of felt design

both qualities. For example, the selection of needle 2 is better because needle 2 is connected positively to the hidden unit. For parameter 3, on the other hand, its low setting is better for them because of its negative connection to the hidden unit. In the same way, it can be seen that the connections with the fourth hidden unit and the seventh hidden unit represent respectively the bad factor for the drainability and the surface smoothness by the negative connections between hidden units and the quality units. The abstract rules expressing how the design conditions affect the qualities can be summarized as follows.

selecting needle 2, low setting of parameter 3,
not selecting treatment 5
=> better for drainability, better for surface smoothness

low setting of parameter 1, high setting of parameter 2,
selecting treatment 2, not selecting treatment 4,
selecting treatment 5
=> worse for drainability

selecting needle 3, high setting of parameter 3,
not selecting treatment 1, selecting treatment 2
=> worse for surface smoothness

It is noted that most of these rules match the intuition of the design experts. The connection structure shows the qualitative relation of

input-output. The quantitative analysis of the relation, on the other hand, should address values of connection weights and unit biases. It might be possible to describe rules associated with the certainty factor calculated from these values, but that is beyond the scope of this paper.

Finally, the distribution factor of the network is increased again to achieve the most refined connections. The network obtained, with a mean-square error of 0.377, is shown in Fig.9. The primal structure of the network is the same as that of the skeleton one. However, one more connection group is created with the eighth hidden unit and some more connections are attached to the existing hidden units. These connections are considered to compensate the details in the representation.

The relation between the learning error and the network's complexity for three results of learning is shown in Fig.10.

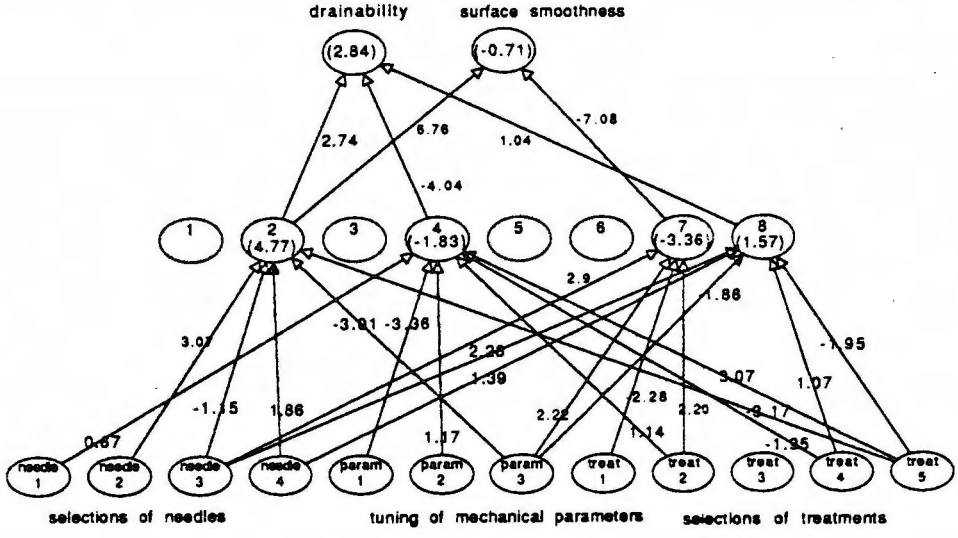


Fig.9 Refined network for application of felt design

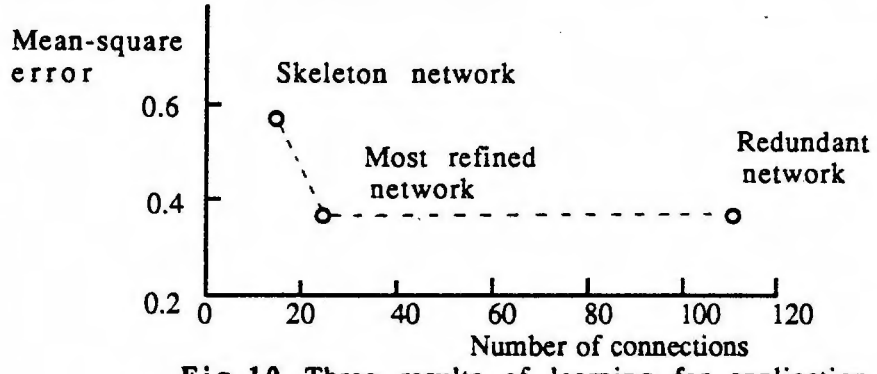


Fig.10 Three results of learning for application of felt design.

5. DISCUSSION

5.1 Generalization in the Proposed Method

Learning is considered to be a process of generating the most generalized rules which explain the relations among the facts shown within the minimum contradictions (Winston, 1970). Inferences to unlearned patterns can be made by interpolating the learned patterns to the unlearned ones along with the generalized rules.

Let us consider the "dogs" and "cats" problem again. Suppose that one more animal appears, whose preference is known to be "fish", but that its cry is unknown. If this new fact is trained by the conventional back-propagation method additionally onto the previously obtained skeleton network of Fig.6, the new connections shown in Fig.11 could be one of the possible results.

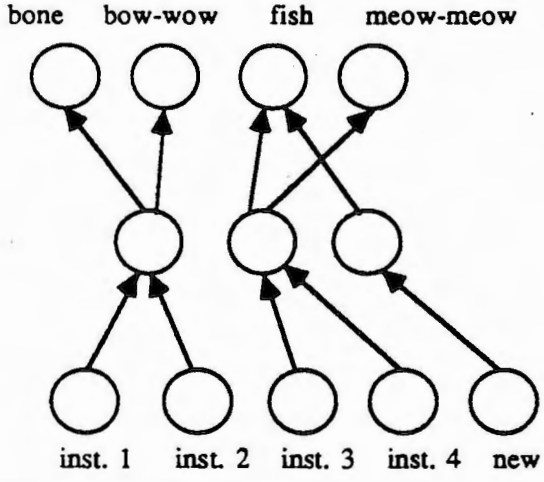


Fig.11 Additional trainning of "dogs and cats" by conventional back-propagation method

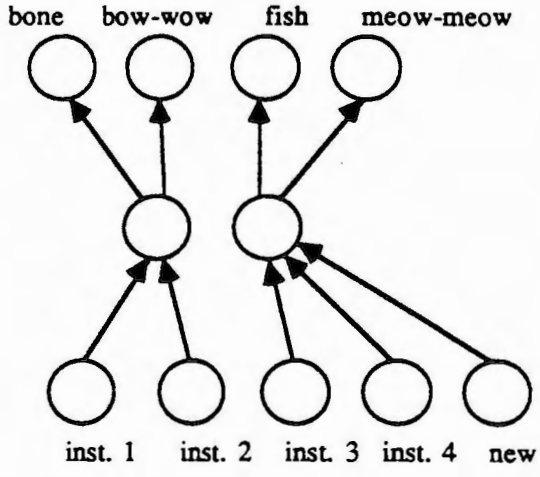


Fig.12 Additional trainning of "dogs and cats" by the proposed learning method

Two additional connections are created, connecting the new animal to "fish" through an additional hidden unit, but not to any units implying its preference. If the same fact is trained by the proposed method, reducing the network's complexity, the network shown in Fig.12 is the only possible result. Only one connection connecting the new animal and the second hidden unit is added. The generalization process forces the new animal to be categorized in the existing schema "cats". This automatically determines the new animal's unknown feature, its cry, as "meow-meow" by the default of the schema.

In the conventional back-propagation method, the pattern of inputs is mapped to that of a smaller number of hidden units. Thus, this

information compression achieves generalization to some extent, but it is weak. The generalization in the proposed method, on the other hand, is achieved positively, accompanied by the reduction of complexity in the connections.

5.2 Why Do We Not Leave Networks as Black Boxes?

A question that might arise is why users need to examine the internal representation of networks. The question addresses the general issue of how much we can trust the results of a neural network's learning. Certainly, neural networks have no a *priori* knowledge to evaluate their own learning. Neural networks can extract as much information as is contained in the data, but no more. If the data used for training is inapposite, networks might learn incorrectly. Thus, it is user's role to make sure that the learning of the network is in the right direction by examining the internal representation of the network obtained.

5.3 Comparison with Other Machine Learning Method

The proposed method is to extract generalized rules from the given examples. The version-space method (Mitchell, 1977), one of the Machine Learning methodology, has been well developed for the same objective. The idea of inductive learning (Michalski, 1983), production of accurate knowledge from the facts, is common to both methodologies, while the approaches seem to differ.

The most distinctive feature of the proposed method is its great flexibility in learning, which is an inherent ability of neural networks. The version-space method requires certain biases which guide searching in the rule space for generalization, while neural networks do not. It can be said, however, that the version space method can utilize a *priori* knowledge for the domain by means of biases, reducing the number of examples for the training. Neural networks maintain freedom in the direction of learning to go, but successful learning requires more examples for the training.

The other distinctive feature of the proposed method is that the method is quite open to the real world environment. The network senses the facts in the real world as an analog pattern, then abstracts them to generalized descriptions in a natural way. In the version-space method, human beings must set up the learning data in a symbolized form. The process at the lowest level is missing. This part of generalizing the real facts to symbolic representation is also part of learning and should be combined in one process so that the whole learning process can proceed adaptively in the real world environment.

6. CONCLUSION

The proposed method derives "skeleton shape" networks representing relations of input-output at the abstract level as well as the refined networks with the minimum redundancy representing up to the specified levels. The method introduces two important factors, explicitness and generalization, to the PDP model in positive attitude. Explicitness is important for users in watching network's learning. Generalization is important for networks in deriving the greatest likelihood for unlearned patterns. In addition, the successful result of application to the design problem suggests that the method could be used as a practical tool for knowledge acquisition.

7. ACKNOWLEDGEMENTS

The authors wish to express their thanks to Chiyoda Corporation for supporting the study and permitting its publication. They also wish to express their appreciation to their departmental manager, T. Takagi for useful discussions.

References

(1) Michalski, R. S., (1983), A Theory and Methodology of Inductive Learning, Machine Learning, Vol. 1. Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds), pp. 83-134, Palo Alto, CA., Tioga Publishing Company.

(2) Minsky, Marvin., (1981), A framework for Representing Knowledge, Mind .Design, J. Haugeland (Ed.), pp. 95-128, Cambridge, MA., MIT Press.

(3) Minsky, M. and Papert, S., (1969), Perceptrons, Cambridge, MA., MIT Press.

(4) Mitchell, T. M., (1977), Version Spaces: A Candidate Elimination Approach to Rule Learning, Fifth International Joint Conference on Artificial Intelligence, Cambridge, MA.

(5) Norman, D. A., Rumerhart, D. E., and the LNR Research Group (1976). Exploration in Cognition, San Francisco, CA., Freeman.

(6) Rumelhart, D. E., McClelland, J. L. and the PDP Research Group, (1986), PARALLEL DISTRIBUTED PROCESSING, VOLS. 1, 2. Cambridge, MA., MIT Press.

(7) Winston, P. H. (1970). Learning Structural Descriptions from Examples. Project MAC TR-76. MIT, Cambridge, MA.

(8) Woods, William A. (1986). Important Issues in Knowledge Representation. Proc. IEEE, 74 (10): 1322-1334.

4th AAAI-Sponsored Knowledge Acquisition for Knowledge-Based Systems Workshop
Banff, Canada, October, 1989.

A Knowledge Acquisition Perspective on Knowledge-Level Models*

Bob Wielinga † Hans Akkermans ‡ Guus Schreiber † John Balder ‡

†Department of Social Science Informatics, University of Amsterdam
Herengracht 196, NL-1016 BS Amsterdam, The Netherlands
Electronic mail: wielinga@swi.psy.uva.nl

‡Netherlands Energy Research Foundation ECN
P.O. Box 1, NL-1755 ZG Petten, The Netherlands
Electronic mail: hans@ecn.UUCP

Abstract

In this paper we investigate the role that knowledge-level models of expert reasoning, knowledge models in short, play in the process of knowledge acquisition. We present a conceptual framework for the specification of knowledge models based on the KADS four-layer model of expert reasoning. This model distinguishes four different types of knowledge: structural –domain dependent– knowledge, knowledge about canonical inferences, knowledge about goals and how to achieve them and strategic knowledge for determining the goal structure of a problem-solving process in a specific problem situation. We show how the KADS four-layer model can be formalized using many-sorted logic and techniques derived from algebraic specification methods. A central role in this formalization is played by our extension to meta-level operations, linking different types and meta-levels of knowledge in a transparent way. Formalized models at the knowledge level may play a role in KBS design that is similar to the conceptual model in traditional information systems development.

1 INTRODUCTION

In the process of acquiring knowledge for a knowledge-based system, a knowledge engineer has to overcome a number of hurdles, such as elicitation, conceptualization and operationalization of the knowledge required to perform a certain task. Although elicitation may be a laborious and time-consuming process in many instances, well-understood techniques exist to support this process. Conceptualization is the process of identifying the relevant conceptual vocabulary in which the knowledge will be expressed, while

*The research reported here was partially carried out in the course of research projects funded by the ESPRIT Programme of the Commission of the European Communities, viz., project numbers 3178 (REFLECT) and 1098 (KADS). The partners in the former project are: UvA, ECN (both NL), GMD and IF (both BRD). The partners in the latter project are STC Technology Ltd., SD-SCICON Ltd., KBSC of TRMC (all UK), NTE Neutech (BRD), Cap Sogeti Innovation (F), and the University of Amsterdam (NL).

operationalization is concerned with the specification and implementation of the representational structure of the system. In current knowledge engineering practice the operationalization activity is often guided by the framework provided by an AI development environment. Well-understood representation techniques such as rules, frames etc. are the basis for this process. The conceptualization of the knowledge supporting a reasoning process, is the activity that mostly lacks support in terms of a coherent framework and methods.

In the KADS (ESPRIT P1098) project a strong emphasis has been placed on developing methods to support this conceptualization process. These methods are based on the notion of a *knowledge-level model* of expertise that defines the different types of knowledge needed for a particular task in an informal way. In this paper we will further investigate the nature of these models, relate them to similar ideas such as *generic tasks*, and show how these knowledge models can be formalized in a logico-mathematical framework. This formalization process is seen here as the intermediate step between conceptualization and operationalization. The aim of the formalization is twofold: first, a formalization clarifies the informal notions that have been used to describe knowledge-level models in the past, and second, a formal specification of knowledge models can be used as a basis for the operationalization of the system.

2 KBS DEVELOPMENT AS MODELING

2.1 The KBS Development Process

In KADS we take the view that the development of knowledge-based systems is essentially a modeling activity [Hayward *et al.*, 1987]. A KBS is not a container filled with knowledge extracted from an expert, but an operational model that exhibits some desired behavior observed or specified in terms of real-world phenomena. The use of models is a means of coping with the complexity of the development process. A model reflects, through abstraction of detail, selected characteristics of the empirical system in the real world that it stands for [DeMarco, 1982].

In KADS, modeling at the knowledge level is an essential intermediate step in the development process [Schreiber *et al.*, 1988]. At the knowledge level one abstracts from the details of the representation to be used in a system. The knowledge model defines the types of knowledge that are to be elicited from the expert and that will have to be represented in the system. In addition, modeling at the knowledge level indicates a route towards the use of generic models for certain classes of tasks, since it is independent of the actual details of the system. These generic task models can be used for top-down knowledge acquisition and system development.

Fig. 1 symbolizes our view of the knowledge engineering process. An observer (knowledge engineer) constructs a conceptual model by abstracting from the problem solving behavior of experts. This abstraction process is aided by the use of some interpretational framework, such as generic models of classes of tasks or of task domains. The conceptual model is a

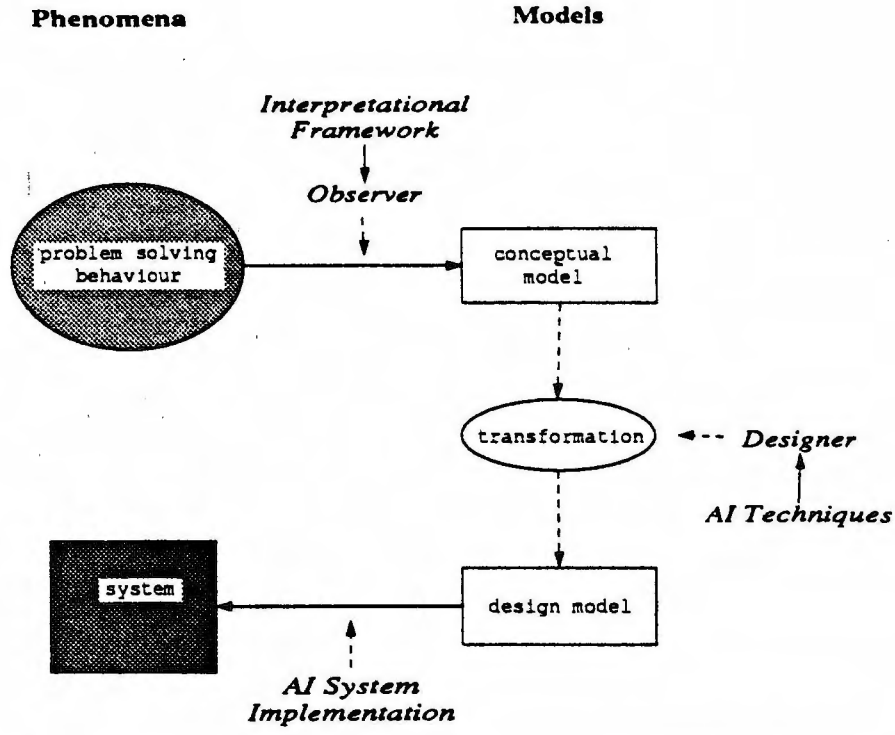


Figure 1: The KBS Development Process: A Bird's Eye View

knowledge-level [Newell, 1982] model of the domain expertise. The conceptual model is real-world oriented in the sense that it is phrased in real-world terminology and can thus be used as a communication vehicle between knowledge engineer and expert. The conceptual model does not take detailed constraints with regard to the artefact into account.

The *design model* is a model of the artefact. It describes how the conceptual model is realized with particular AI techniques and methods. The idea is that the design model can be transformed into a detailed system design and subsequently into actual *system code* without further major decisions having to be made.

We have experienced that building a conceptual model model without having to worry about system requirements makes life easier for the knowledge engineer. The distinction between the two models is also useful, if one opts for another problem-solving method in the artefact than the one used by the domain expert. For example, if the domain expert uses heuristics to find a particular solution, one may choose to use a exhaustive *generate and test* method in the KBS, because its computational resources are larger. An explicit representation of both models can in that case be used to explain the differences in behavior between the system and the expert to a user.

2.2 A Four-Layer Conceptual Model

In earlier work [Wielinga & Breuker, 1984; Wielinga & Breuker, 1986] we proposed a framework for modeling expertise, based on a categorization of knowledge according to the role that it plays in a reasoning process. In this framework four categories of knowledge are distinguished.

The first category of knowledge concerns *static domain knowledge*. The static domain knowledge consists of the domain concepts and their attributes, domain facts, structures representing complex relations etc. The static knowledge can be viewed as the layer of knowledge representing a declarative theory of the domain. In fact, adding a simple deductive capability would enable a system in theory, but not in practice, to solve all problems solvable by the theory. The static domain knowledge is considered to be largely task neutral, i.e. represented in a form that is independent of its use. There is ample evidence [Wielinga & Bredeweg, 1988] that experts are able to use their domain knowledge in a variety of ways, e.g. for problem solving, explanation, teaching etc. Separating static domain knowledge embodying the theory of the domain from its use in a problem solving process, is a first step towards flexible use and reusability of domain knowledge.

A second type of knowledge concerns canonical inference steps. An inference is considered to be an elementary step in the reasoning process deriving new information from existing information. An important characteristic of an inference is the type of information that is used in making the inference and the type of information that it produces. In addition, the inference is characterized by the type of domain concepts and relations it uses. For example, an elementary classification inference takes as input a bundle of features of an object and produces as output a class to which the object can belong, using definitions of the set of classes that the system knows about. The information used or produced by a canonical inference is typed by a *metaclass*. In the terminology of the KADS four-layer model inferences are called *knowledge sources*.

With respect to the question of a taxonomy of canonical inferences there is no consensus in the literature. The most well known collection of canonical inference steps can be found in Clancey's model of heuristic classification [Clancey, 1985]. In this model the inferences are: *abstraction*, *association* and *refinement*. Within our model we assume a somewhat larger set of canonical inferences. Example additions are *transformation*, *selection* and *computation* (a full description of this set can be found in [Breuker *et al.*, 1987]). If canonical inferences are viewed as functions, one is led to interpret metaclasses as related to the arguments of these functions (this intuitive idea will be made precise in Sec. 4). A metaclass describes the role a group of domain concepts can play in the reasoning process (*e.g.*, datum, feature, hypothesis, solution).

The third category contains knowledge about how elementary inferences can be combined to reach a certain goal. The prime knowledge type in this category is the *task*. A task can be defined statically (a control structure with the usual control primitives) or dynamically (as a planner function which interprets descriptions of the second category). Tasks satisfy a particular *goal*. The relations between tasks and goals are in principle many-to-many. Task knowledge is usually characterized by a vocabulary of control terms, for instance

indicating that a finding has been processed or a hypothesis has been verified.

The fourth category of knowledge is the *strategic knowledge*. Strategic knowledge determines what goals are relevant to solve a particular problem. How each goal is achieved is determined by the task knowledge. Strategic knowledge will also have to deal with situations where the afore-mentioned knowledge categories fail to produce a partial solution. For example, the problem-solving process may reach an impasse because information is not available or because contradictory information arises. In such cases the strategic reasoning should suggest new lines of approach or attempt to introduce new information *e.g.*, through assumptions (cf. [Jansweijer, 1988]).

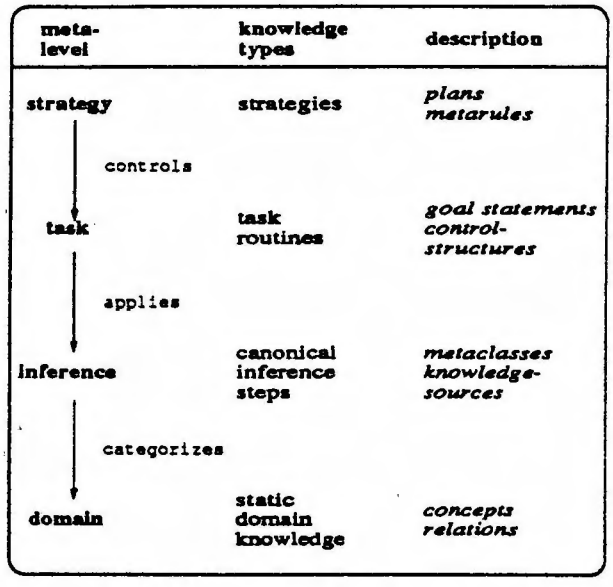


Figure 2: The KADS four-layer model

These categories can be seen as knowledge *layers* in the sense that each successive layer interprets the description at the lower layer. In Fig. 2 the four categories and their interrelations are summarized. The four-layer framework for knowledge modeling has been successfully used as a basis for structured acquisition and description of knowledge at an intermediate level between the verbal data obtained from experts and the knowledge representation in an implemented system [deGreef & Breuker, 1985].

From a knowledge-level viewpoint, the present four-layer model captures knowledge categories that are quite similar to those encountered in other models in the literature. However, differences exist about where to situate particular types of knowledge. This point will be further discussed in the next section. An important problem common to all approaches is that they are formulated in a rather informal, and therefore sometimes ambiguous, manner. Accordingly, in Sec. 4 we will construct a formal framework for the specification of knowledge models that will allow us to explore some questions in more depth.

3 KNOWLEDGE MODELS IN PERSPECTIVE

One major objection to the use of knowledge-level models in the KBS development process is the potential computational inadequacy of such models. Since knowledge-level models do not specify the control regime in full detail they are apt to potential combinatorial explosion behavior. Although it is true in principle that this problem may occur, the structure of the models proposed here provides important safeguards against the computational inadequacy. The knowledge that is specified in a KADS four-layer model cannot be used in arbitrary ways, it has to fulfill certain typing requirements and can only be applied within the constraints specified by the model. For instance, a logical implication of a certain type can only be used for abstraction inferences, not for all types of inference steps. Our introduction of knowledge sources and task structures generally yields the possibility of selecting specific rules or theories needed to produce a certain inference. This is also practically shown in our example specification of Sec. 5. In this way the knowledge-level model provides a *role-limiting* [McDermott, 1989] constraint to knowledge, and hence reduces the danger of a combinatorial explosion.

Some recent approaches to knowledge-based system development take a similar viewpoint as KADS. However, the terminology and classification of knowledge types differ. Alexander [Alexander *et al.*, 1988] for example, defines the static ontology as one of the essential parts of a knowledge level model. A static ontology defines the well-formed sentences that can be formulated in a particular domain. The static ontology is in many respects similar to the KADS domain layer, albeit that KADS limits the conceptual primitives at the domain layer largely to concepts and relations.

The work of Clancey [Clancey, 1985] on heuristic classification is clearly a precursor of the KADS four-layer model. Clancey argues that many classification tasks can be described at the knowledge level in terms of *abstraction* steps, abstracting data to abstract problem features, *association* steps between abstracted data and abstract solutions, and *refinement* steps of the abstract solution to a specific solution. The heuristic-classification model does not, however, specify the nature of the domain knowledge that is needed to support the inferences. In addition to the inference structure Clancey defines problem solving behavior in terms of *tasks* which are procedures to achieve a particular goal. In a similar spirit, Steels [Steels, 1988] distinguishes the following types of knowledge underlying expertise: domain models, describing what form domain concepts and facts can take; role annotations, describing what role domain knowledge can or should play; task knowledge, describing the structure of the problem-solving task; problem-solving methods, indicating how a task may be performed; and strategic knowledge.

Although terminology is different, a common view appears to be emerging based on the idea that different types of knowledge constitute the knowledge level and that these different types of knowledge play different roles in the reasoning process and have inherently different structuring principles. One salient characteristic is that all approaches distinguish between structural domain knowledge and control knowledge. In addition, various kinds of control knowledge are distinguished, like global control of how to go about the task as a whole, and local control knowledge specifying how and/or when to carry out certain individual actions.

The KADS four-layer model and our notion of interpretation models (inference structures that are typical for certain classes of problems [Breuker *et al.*, 1987]) provides a framework to further investigate the notion of *generic task*. Chandrasekaran [Chandrasekaran, 1988] defines a generic task as building blocks of reasoning strategies that are generic and can be used as components for complex reasoning tasks. It appears that generic tasks are very similar to the interpretation models used in KADS.

Similarly, [Eshelman *et al.*, 1988] defines the generic method *cover and differentiate* for diagnosis in terms of the following steps:

1. Determine events that potentially explain symptoms.
2. Identify information that can differentiate between candidate explanations by ruling out, providing support for candidates, providing preferences.
3. Get this differentiating information and apply it.
4. If new symptoms become available, go to step 1.

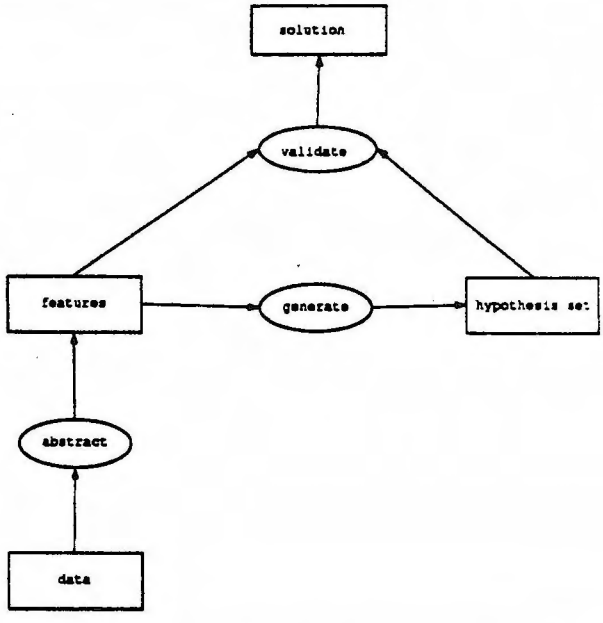


Figure 3: Inference structure for cover-and-differentiate

In this example, several types of knowledge are specified in an informal way. First, the domain should provide concepts like: event, symptom, explanation link, preference, rule-out relations etc. Second, a number of basic inference types (similar to knowledge sources) are defined: generating a hypothesis, matching an hypothesis to the available data, selection or ordering, compute preferences. Fig. 3 shows a somewhat simplified version of the cover-and-differentiate method. Our example discussed in Sec. 5 gives a full formal specification of this generic inference structure and shows how it applies to a simple medical domain.

In addition there is control knowledge that indicates that all possible candidates are generated given a set of symptoms, and that differentiating information is obtained in a backward manner. These types of knowledge would be located in the task layer of KADS.

There is no doubt that generic models of classes of tasks can be a great help in knowledge acquisition, as systems like MOLE show, but there is little consensus in the literature about what constitutes a generic model. The KADS framework provides a handle on some of the issues that arise from the current literature on generic tasks, but the notions like knowledge source, meta class and task structure are too informal to give complete clarity. This problem of the informality of concepts at the knowledge level is a general one in the literature, and this is likely to constitute a major obstacle to solving the outstanding issues we have pointed at. In the following sections we will therefore develop a framework for a more formal specification of knowledge-level models.

4 OUTLINE OF A FORMALIZATION OF KNOWLEDGE MODELS

A major outcome of the knowledge-acquisition process will be an intermediate description of the considered expertise that serves as a basis for developing the knowledge-based system. In first-generation expert systems such a description is usually embodied in a prototype. Here, the model of knowledge may be said to be formal (in the sense of being executable), albeit a symbol-level model only. In contrast, the KADS methodology generates knowledge-level models of expertise which are conceptual in nature. KADS enables to describe a body of knowledge at a much higher, for the domain expert more 'natural', level of aggregation as well as from multiple points of view (cf. Figs. 1 and 2. This greatly facilitates the communication about and the transfer of knowledge. More generally, it is noted that an intermediate high-level description of the expertise is already valuable in itself for many organizations, apart from some representation in a knowledge-based system. Admittedly, the KADS knowledge models are informal. This limits their conceptual precision and their empirical verifiability and complicates their translation to a system design and implementation.

Thus, one would like to have the best of both worlds. On the one hand, the high-level conceptual vocabulary introduced by methodologies like KADS appears to be very useful as well as flexible in the analysis and structuring of knowledge. On the other hand, the ensuing knowledge models should preferably be formal (in the sense of mathematics or logic). Such a formalization would represent a significant help in guiding the knowledge engineer's activity. Accordingly, we will outline below the various ingredients of a general approach to the *formalization of models of expertise at the knowledge level* [Akkermans *et al.*, 1989a; Akkermans *et al.*, 1989b].

4.1 Knowledge Modules

An intuitive notion used in analyzing knowledge-acquisition data is that of *knowledge structures*. Knowledge is viewed as decomposable into more elementary units – to be called here knowledge modules. By linking these modules in certain ways one can construct more complex knowledge structures. We make no statements as to the 'grain size' of the knowledge modules: it depends upon the domain of expertise and on the aims of the

knowledge analysis and the intended KBS, and is left at the discretion of the knowledge engineer (this is no difference from modeling practices in other scientific and engineering disciplines). It may be stated, however, that a symbol-level approach like the common prototyping tends to take too microscopic a view, whereas the 'coarse-grained' perspective introduced by KADS is more adequate for the purposes of knowledge acquisition. We postulate a ***knowledge module*** to be given by a simple many-sorted logico-mathematical structure (called a conceptualization in [Genesereth & Nilsson, 1987]), *i.e.*, a sequence of a set of sorts (the universe), a set of constants (typed individual objects) and sets of functions and relations defined on these objects. By analogy with an algebraic specification, we define a formal specification of a knowledge module M to be the pair:

$$Spec(M) = \langle \Sigma, B \rangle.$$

Here, Σ is the signature. This set contains the names of the sorts, individuals, functions and relations occurring in the logical structure (conceptualization) represented in the knowledge module. The signature generates the alphabet for a logical language, which we take to be here a many-sorted predicate logic (although other choices are possible), including some complete deduction system. B is a set of axioms in this language, modeling the aspects of expert knowledge under consideration. We remark that this determines the semantics of specifications in line with our overall view of knowledge engineering as the modeling of specified expertise on the basis of empirical knowledge-acquisition data: throughout this work we employ an intended-interpretation semantics.

The above is called a 'flat' specification in the algebraic terminology. As an algebra is a structure without relations, the above definition is slightly more general than an algebraic specification. Admitting relations, however, leads in our opinion to a somewhat more natural way of expressing knowledge-level models.

4.2 Module Algebra

The specification of knowledge modules constitutes the first step in the formalization of knowledge-level models. The second step is the formal construction of more complex structures out of the modules. As a partial solution to this problem, we suggest to employ the ***module algebra*** as developed in algebraic-specification theory [Bergstra *et al.*, 1989]. With the aid of special set-theoretic operators this algebra forms new, composite (*i.e.*, non-flat) modules from existing ones. Thus, we are dealing with a higher level of aggregation (and this is precisely the advantage of using module algebra at the knowledge level): the individual domain objects of this algebra are the modules. The main operators of this algebra are *IMPORT* and *EXPORT*. The *IMPORT* operation is the basic means for combining two modules into a single larger one, by taking the set union of their signatures and of their axioms. The *EXPORT* operator splits the signature of a module into a (for other modules) visible and a hidden part. It may thus be seen as a formalization of the conventional data abstraction or 'information hiding'. Other useful module-algebraic operators are *PARAMETRIZATION* and *RENAMING*. A number of variations and extensions exist or are possible with respect to this basic scheme of module algebra.

4.3 Specifying Meta-Level Knowledge: the Lifting Concept

We believe that the above aspects, *viz.*, knowledge-module specifications and module algebra, already constitute a significant step forward towards a formal description of expertise at the knowledge level. A practical syntactical format for such a description may be obtained by slightly modifying existing algebraic specification languages [Goguen & Meseguer, 1987; Jonkers, 1988]. Nevertheless, there is one important aspect of expertise that is difficult to model within this approach, namely, meta-level reasoning. The standard operators of the module algebra are useful for introducing and expressing structure *within* a single level (or KADS layer). They are of no help, however, in answering questions concerning, for example, the relation between domain concepts and the associated meta-classes, knowledge sources or task structures.

In order to be able to switch between meta-levels, we need to extend the module algebra with new *meta-level operators*. For obvious reasons we will call these *LIFT* operators. The typical *schema* of the *LIFT* operation is given by a more explicit notation of existing ideas on meta-level reasoning [Weyhrauch, 1980; Genesereth & Nilsson, 1987]:

$$LIFT(\alpha) = `\alpha',$$

where α represents a relation or axiom (thus, an element of the signature or of the theory of a module) at the object level, and 'α' denotes a corresponding object constant (of a certain sort) at the meta-level. The *LIFT* operation is a representation of what is sometimes also called a naming relation between object and meta levels. In some cases it will be advantageous to employ extensions of the above definition, by mapping onto closed terms of the meta-level in general.

This *LIFT* operation provides a basic ingredient for specifying the connection, by means of a function yielding a one-to-one correspondence, between (subsets of) the object- and meta-levels. More specifically, this formal concept is helpful to clarify some aspects of the KADS methodology. For instance, it is seen that the domain concepts at the domain layer and the corresponding metaclasses of the inference layer are related precisely, the metaclasses are formally interpreted here as the sorts associated with the inference-level objects that are output from a *LIFT* mapping going from the domain layer to the inference layer. This formal notion fits quite well with the earlier KADS concept of metaclasses as the argument roles of domain concepts at the inference level [Wielinga & Breuker, 1986].

As noticed, the *LIFT* mapping is a crucial element in switching to a meta-level. It is to be pointed out, however, that it does not represent the whole story, since it only generates meta-level objects (variable-free terms). It is possible to extend this to a complete (what we will call a) 'lift module' which also contains basic meta-statements of the type: this object-level axiom is called a definition, is an implication sentence, is a domain fact, *et cetera*. Then, the completed lift module may be modularly structured with the aid of the discussed algebra like any other module. Next, the lift module (more precisely, a reduction of it to a smaller set of axioms) may be imported into a knowledge source. The latter is interpreted in the present context as a generic meta-level theory and, hence, it is specified as a generic (possibly parametrized) meta-level knowledge module. Accordingly, a knowledge source has a pure meta-level character, whereas a lift module has a typical

interface nature, in other words, it 'sees' both the object-level and meta-level sides. The object-level aspects can be hidden from the meta-level modules through the application of module-algebraic operations to the lift module, like export of a limited signature and reduction of the set of axioms. The reason to do so is to maintain a clear separation between the two levels. It is therefore also implied that deductive steps within the context of a lift module are restricted to those that have a translation (causal connection) nature between the object- and meta-levels.

This whole process – performing a *LIFT* operation, extending this to a completed lift module, hiding the object-level part of the signature and restricting the module to the relevant meta-level axioms – is in its entirety denoted here as the 'lifting concept'.

Discussion This completes our generic ideas regarding formalization of the knowledge level. They will be illustrated and be rendered more concrete through an example to be presented in the next section. We point out that, although independently developed here, the use of techniques derived from many-sorted algebraic specifications is not completely new. They are also used in the ontological-analysis approach of [Alexander *et al.*, 1988]. It appears, however, that the power of these techniques is not yet fully recognized in the context of knowledge acquisition and expert systems building. Also, the specification vocabulary seems in the present work to be worked out in more and richer detail, including the proposed use of module algebra. Furthermore, the general scheme of the *LIFT* mapping is well known [Genesereth & Nilsson, 1987], but we note that its expression via the suggested specification methods and its embedding in a module algebra (the resulting whole designated as the 'lifting concept') yields a framework that is more explicit and flexible than usual. Generally speaking, the lifting concept provides a *conceptual* abstraction mechanism, distinct from the usual data abstraction, whereas it can be treated on the same footing as other knowledge modules.

In sum, the combination of existing elements, *viz.*, (i) the module specification techniques and the module algebra; (ii) the lifting concept; and (iii) the conceptual framework of KADS, and their application to knowledge acquisition seems to us a relevant advance. From a knowledge-level perspective, the formalism introduced above brings about several advantages. First of all, given the fact that the purely conceptual framework has shown in practice to give useful guidance to knowledge engineers, a formal framework will be even more helpful in this respect. Second, the structured conceptualization of a domain is supported by and can be clearly expressed in terms of sorts, constants, predicates and operators. Moreover, it is open to gradual refinement and may thus be of help in high-level prototyping. Third, the framework facilitates a succinct indication of the types of knowledge by means of the sorts in each substructure. Fourth, the axioms in the module specifications form a natural way of expressing factual and control knowledge, without the burdens of representational and implementational issues. Finally, the notion of lifting naturally corresponds to assigning certain roles to knowledge elements of lower levels at higher levels.

5 EXAMPLE SPECIFICATION OF A KNOWLEDGE MODEL

In this section we present a practical example of a knowledge-level specification according to the formalism outlined in the previous section, as applied to the knowledge layers of KADS. The example concerns the early diagnosis of coronary diseases (albeit for the present purpose in a rather stylized and idealized form), and has been derived from an actual, though prototypical, implementation. The main purpose of this section is to show that this formalism is able to express high-level knowledge types in a generic and structured manner. The notation used below for the knowledge specifications is adapted from algebraic specification languages such as OBJ2 [Futatsugi *et al.*, 1985] and COLD [Jonkers, 1988].

5.1 Formal Specification of Domain Layer Knowledge

The two modules below specify which patient data may be used in the reasoning process. We assume throughout the specifications that primitive types like the natural numbers and the booleans have been specified and imported.

module *patient*
  sorts
    object
  functions
    systolic_bp: object → nat
    diastolic_bp: object → nat
    cigarettes: object → nat
    age: object → nat

module *patient_with_pain*
  import
    patient
  sorts
    duration, pain
  constants
    constantly, long, short: duration
    mid_sternal_pain: pain
  variables
    object: X
    pain: P
  functions
    has_pain: object × pain → bool
    time_interval: object × pain → duration
    stress_triggered: object × pain → bool
  axioms
    stress_triggered(X,P) ∨ $\exists_d$ time_interval(X,P) = d ⟹ has_pain(X,P)

The above axiom specifies a (for humans quite trivial, but not for computers) constraint: if the relevant symptom characteristics are observed, then so is the symptom itself. Later on, this axiom will in addition be used to abstract from these specific characteristics and to enable to consider the symptom only.

The domain theory is further represented in a similar fashion as in NEOMYCIN: the module *domain_theory* is constructed (via *IMPORT*) from three other modules: abstractions, refinements, and associations:

module *domain_theory*
  import
    data_abstractions, associations, solution_refinements

In *data_abstractions* abstraction rules are defined from domain data to domain features for our example:

module *data_abstractions*
  import
    patient
  variables
    object: X
  relations
    diabetic, hypercholestorolaemy, high_risk_person, hypertensia, heavy_smoker,
    middle_aged: object
  axioms
    diastolic_bp(X) $\geq$ 90 $\Longrightarrow$ hypertensia(X)
    cigarettes(X) $\geq$ 20 $\Longrightarrow$ heavy_smoker(X)
    age(X) $\geq$ 40 $\Longrightarrow$ middle_aged(X)
    middle_aged(X) $\wedge$ heavy_smoker(X) $\wedge$ hypertensia(X) $\Longrightarrow$ high_risk_person(X)
    diabetic(X) $\vee$ hypercholestorolaemy(X) $\Longrightarrow$ high_risk_person(X)

Solution_refinements contains two kinds of implications. The first axiom defines a partial order on the predicates (classes) specified in this module. The other two axioms represent distinctive properties of a class. To keep the example simple, this module will not be actually used in the meta-layers discussed below: only the reasoning concerning the intermediate hypothesis of *coronary_disease* will be fully specified.

module *solution_refinements*
  import
    patient_with_pain
  variables
    object: X
  relations
    coronary_disease, angina_pectoris, myocardial_infarction: object
  axioms

angina_pectoris(X) ∨ myocardial_infarction(X) ⟹ coronary_disease(X)

angina_pectoris(X) ⟹ time_interval(X.mid_sternal_pain) = short ∧
stress_triggered(X.mid_sternal_pain) = true

myocardial_infarction(X) ⟹ time_interval(X.mid_sternal_pain) = constantly ∧
stress_triggered(X.mid_sternal_pain) = false

Associations relate abstracted data (*i.e.*, features) to possible solution classes.

module *associations*
import
data_abstractions, solution_refinements
axioms
high_risk_person(X) ∧ has_pain(X.mid_sternal_pain) ⟹ coronary_disease(X)
< other associations >

The main advantage of the specification approach taken here is that the knowledge types as well as the structure of the domain knowledge have become transparent. It is clear which functions and relations are used, what data can occur in the database and what further knowledge will have to be acquired from an expert. The specification constitutes in effect a static domain model of what a system will be able to reason about, and what it will ignore.

5.2 Inference Layer Specification

The domain modules described above are the basis for the specification of a reasoning system. In the second layer we specify what generic inferences can be made on the basis of this domain knowledge and what the dependencies between these inferences are. In order to obtain a domain-independent description at the inference layer, one has to carry out a conceptual abstraction step from the domain level to the inference level. This is done by means of the LIFT operator which provides a mapping between domain predicates and sentences of a certain type to constants or closed terms at the next level. The LIFT operator can be realized in a system by a number of rewrite rules. To the LIFT mapping axioms (which can be viewed as a causal connection specification and, thus, have a mixed-level character) we add meta-level predicates and associated axioms which may be used by the knowledge source modules of the inference layer.

The lift module between the domain and inference layers now becomes:

module *lift1(domain_layer → inference_layer)*
import
patient_with_pain, data_abstractions, associations
sorts
data, features, solution_class
constants

$d_{i,i\in nat}$: data
f_i: features
s_i: solution_class

variables

D_i: data
F_i: features
E: data ∪ features
S_i: solution_class

relations

provable: data ∪ features
abstraction_theorem: $(\text{data})^n \times \text{features}$
redflag_finding: features
association_theorem: $(\text{features})^n \times \text{solution_class}$

upward axioms

axiom scheme for: $\forall p_i \Longrightarrow q_i$ from patient_with_pain and data_abstractions:
$\text{LIFT}(p_i) = d_i \; ; \; \text{LIFT}(q_i) = f_i$

downward axioms

$\text{has_pain}() = \text{LIFT}^{-1}(f_1)$
$\text{high_risk_person}() = \text{LIFT}^{-1}(f_5)$
$\text{coronary_disease}() = \text{LIFT}^{-1}(s_1)$
<first axiom from associations> $= \text{LIFT}^{-1}([f_1, f_5, s_1])$
$\text{provable}(E) \Longleftrightarrow \text{LIFT}^{-1}(E) = \text{true}$

meta axioms

$\text{redflag_finding}(f_1)$
$\text{abstraction_theorem}(d_i, f_i) \; . \; i = 1, 2, \ldots$
$\text{association_theorem}(f_1, f_5, s_1)$

On the basis of this lift specification, one can at the inference meta-level distinguish between types of domain sentences, *e.g.*, data or feature type sentences. The upward and downward axioms intend to represent the causal connections between the layers, while the meta-axioms are expressed in the meta-language only and may be imported into the knowledge sources of the inference layer. The *provable* predicate is interesting here, because its definition entails a downward connection, whereas LIFT represents an upward connection. This suggests how a reflective connection may be specified at the knowledge level [Akkermans *et al.*, 1989a; Akkermans *et al.*, 1989b], although here rather rudimentarily so. We will discuss reflective aspects in a forthcoming, separate publication.

With the help of this lift module, we can now specify the canonical inferences (called knowledge sources in KADS) in a concise and generic way. These inference modules are given below. The overall inference structure is depicted in Fig. 3.

First, we specify the generic abstractions that can be made in this domain:

module *feature_abstraction*

import

lift1

axioms

$\text{abstraction_theorem}(D,F) \wedge \text{provable}(D) \Longrightarrow \text{provable}(F)$

It is to be noted here that, for convenience, we have lumped various abstraction methods into a single one, indicated by the predicate *abstraction_theorem*. This can be seen from the fact that in the lift1 module we only have employed a single axiom scheme for lifting the domain axioms that carry out abstraction type activities. In a real case, one would probably distinguish between different kinds of abstraction theorems (*e.g.*, qualitative, syndrome and generalization abstractions following the terminology of Clancey). The introduction of these distinctions is possible through employing different axiom schemes and, thus, presents no problems for our formalism, apart from extending the specification.

The next module shows how hypotheses (candidate solutions) can be generated on the basis of the features that have been abstracted from the data:

module *hypothesis_generation*
  import
    feature_abstraction
  relations
    hypothesis: solution_class
  axioms
    association_theorem($F_1, ..., F_n, S$) $\wedge$
    $\exists F_i$ (redflag_finding(F_i) $\wedge$ provable(F_i))
    $\Longrightarrow$ hypothesis(S)

Finally, each generated hypothesis needs to be validated by showing that it actually covers *all* relevant features (symptoms); the hypotheses that pass this test are acceptable solutions:

module *solution_validation*
  import
    hypothesis_generation
  relations
    solution: solution_class
  axioms
    hypothesis(S) $\wedge$ association_theorem($F_1, ..., F_n, S$) $\wedge$
    $\forall F_i$ provable(F_i)
    $\Longrightarrow$ solution(S)

Together, these modules give a precise and formal specification of a (somewhat simplified) version of McDermott's generic *cover and differentiate* method, as discussed in Sec. 3.

5.3 Task Layer Knowledge

At the task layer we are concerned with *goals*, *i.e*, with control aspects related to operations (knowledge sources) at the inference level. For example, when the abstraction knowledge

source is activated from a certain task, this may result in a new fact that has been derived, or in one or more subgoals that are needed to make the abstraction. The task layer obtains its information about the results of the reasoning at the inference layer through a LIFT operation. The task layer may also need to ask certain information from the inference layer, *e.g.*, whether a certain feature can be obtained through abstraction. This entails a form of the inverse LIFT operation, which we have already seen. Hence, the relation between the task layer and the inference layer is a reflective one.

We will focus below on a practical task-structure specification that is made possible by the present inference structure (Fig. 3). First, however, we will describe the lift connection between the inference and task layers.

module *lift2(inference_layer → task_layer)*
import
feature_abstraction, hypothesis_generation, solution_validation
sorts
metaclass, element, knowledge_source
constants
data_set, feature_set, hypothesis_set, solution_set: metaclass
$\delta_{i, i \in nat}$: element
abstract_ks, generate_ks, validate_ks: knowledge_source
variables
$Metaclass$: metaclass ; E_i: element; KS: knowledge_source
relations
obtain: metaclass
nonempty: metaclass
apply: knowledge_source × metaclass
upward axioms
LIFT(<axioms from feature_abstraction>) = abstract_ks
LIFT(<axioms from hypothesis_generation>) = generate_ks
LIFT(<axioms from solution_validation>) = validate_ks
LIFT(provable(d_i)) = δ_i
$\bigwedge_i$ LIFT(provable(d_i)) = $[\delta_1, ..., \delta_n]$
LIFT($\bigwedge_i$ provable(d_i)) = data_set
< analogous axioms for feature_set, hypothesis_set and solution_set,
in relation to the predicates provable, hypothesis and solution, respectively >
downward axioms
obtain($Metaclass$) $\Longleftrightarrow$ LIFT^{-1}($[E_1, ..., E_n]$) = true
nonempty($Metaclass$) $\Longleftrightarrow$ $\exists E_i$ LIFT^{-1}(E_i) = true
apply(KS, $Metaclass$) $\Longrightarrow$ LIFT^{-1}(KS) $\wedge$ $\bigwedge_i$ (LIFT^{-1}(E_i))
meta axioms
data_set = $[\delta_1, ..., \delta_n]$
< analogous axioms for feature_set, hypothesis_set and solution_set >

This lift module enables us to speak at the task layer about knowledge sources (theories of the inference layer) and metaclasses (sets at the inference layer) as a whole. The meta-axioms only have an auxiliary nature here, identifying a list of items with a single set name,

thus enabling a compact way of expression. The specification also gives the possibility to inspect these sets via the list of the elements they contain. This is useful for certain task checks, *e.g.*, making sure that there really are sensible hypotheses available on the basis of actual valid data, before trying to verify hypotheses. The *obtain* predicate plays a downward reflection role at the task layer, similar to the role of the *provable* predicate at the inference layer. Because these predicates are coupled according to the above lifting axioms, their combined effect is that of a doubly downward connection. In this way, the task layer can obtain information about the actual patient data from the domain layer.

We are now able to specify various tasks at the KADS task meta-level. The overall task that employs and controls the whole of the inference structure of figure 3, can be intuitively formulated as the following sequence of steps (*i.e.*, subtasks). After having collected the actual patient data, abstract from them the relevant symptom features. Then, generate on their basis the appropriate hypotheses and, next, validate these. The end result will be a set of (in the context of the given data and the presently specified theory) acceptable solutions. This is formally specified below.

module *find_solutions_from_data_task*
  import
    lift2
  variables
    Data_set, Feature_set, Hypothesis_set, Solution_set: metaclass
  relations
    abstract: metaclass × metaclass
    generate: metaclass × metaclass
    validate: metaclass × metaclass
  axioms
    obtain(Data_set)
    abstract(Feature_set, Data_set) ⟸
        nonempty(Data_set) ∧
        apply(abstract_ks, Data_set) ∧
        obtain(Feature_set)
    generate(Hypothesis_set, Feature_set) ⟸
        nonempty(Feature_set) ∧
        apply(generate_ks, Feature_set) ∧
        obtain(Hypothesis_set)
    validate(Solution_set, Hypothesis_set) ⟸
        nonempty(Hypothesis_set) ∧
        apply(validate_ks, Hypothesis_set) ∧
        obtain(Solution_set)

It is gratifying that, after all the switching between meta-levels, this formal specification is still so close to the informal intuition related to this task structure. The predicates *obtain*, *abstract*, *generate*, and *validate*, may each be viewed as representing a task on itself, as they are linked to activating a single knowledge source. This is in line with the earlier informal KADS view that the execution of knowledge sources is the most elementary task statement occurring in a task structure.

In the above task specification also some simple checks have been introduced via the *nonempty* predicate concerning the usefulness of undertaking subsequent task steps. Actually, these checks are to be seen as 'strategic handles', in the sense that upon failure of such checks it follows that the task as a whole cannot be successfully carried out and, hence, a new strategy is needed. What should be done in this case, needs to be specified in the strategy meta-level, the fourth layer of KADS. It is not difficult to give the strategy layer relating to the present example: *top goal* is to execute the axioms of the task module (lifted to some object constant at the strategy level); *exit* if something fails along the road (for example, due to some set being empty; given the fact that there are no alternative strategies to overcome such failures in the present example, there is no other strategic choice than exiting).

Finally, we notice that the *apply* predicate selects and restricts the application of specific knowledge sources to specific metaclasses. This is an indication that a knowledge-level model specification can be a role-limiting constraint regarding the combinatorial explosion. In this way the above knowledge model can give high-level support in relation to the computational adequacy of a knowledge-based system, a problem discussed in general terms in Sec. 3.

In sum, through the above example we believe to have made plausible that a formal, and yet flexible and rather natural, specification of a large variety of high-level knowledge types in a single, unified framework is possible.

6 CONCLUSIONS

The fundamental premise of this paper is that knowledge-level models are a suitable intermediate representation of the knowledge that needs to be acquired for a knowledge-based system. The knowledge model acts as a high-level specification for the system to be built, in much the same way as a conceptual model is a specification of a conventional information processing system. The major ingredients of a knowledge model appear to be: static domain ontology, inference types and the roles that domain structures can play in the reasoning process, the control structures at the task and strategic levels. These four types of knowledge are all explicitly represented in the conceptual framework of KADS, discussed in Sec. 2, but they form a continuing theme in the many different formulations of knowledge-level models.

In Sec. 3 we have presented our points of view on various significant issues in the literature pertaining to knowledge-level modeling. The knowledge-level model should be distinguished from the actual representation and implementation methods that are used to realize a certain behavior in a computational system, (Fig. 1). To make this aspect more explicit, we propose to distinguish between the models of empirical expertise (called a conceptual model) and high-level models of the artefact (called a KBS design model here).

Although the conceptual knowledge-level model should be plausible with respect to computational adequacy, it abstracts from the details of a particular method or control regime.

However, it does limit the role that pieces of knowledge can play in a reasoning process and as such it has a *role limiting* function that constrains the combinatorial explosion. This point can be clearly seen in our example specification in Sec. 5.

Several authors have proposed generic problem-solving methods for certain classes of tasks. It is evident that such models can play a crucial role in knowledge acquisition, but the precise nature of such models is as yet not clearly defined in the existing literature. The separation of inference and control knowledge from the domain knowledge in the KADS four-layer model, facilitates the definition of knowledge models for generic tasks abstracted from a particular domain. Also this point is exemplified in Sec. 5.

We have demonstrated (Sec. 4) how the KADS conceptual framework for the knowledge-level modeling of expertise can be formalized. This is achieved through an extension of methods from many-sorted algebraic specification and the associated module algebra. The specification format of knowledge models can be adapted from existing algebraic specification languages, as shown by example in Sec. 5. The main extension lies in our *lifting concept*, introducing additional meta-level construction operators, which can be transparently treated in the same specification format. This lifting concept provides a *conceptual abstraction* mechanism that is crucial for the modeling of high-level knowledge types. We mention in passing that this approach can be extended to the description at the knowledge level of reflective reasoning [Akkermans *et al.*, 1989a; Akkermans *et al.*, 1989b]. We will discuss this problem in a future publication.

The formalized, yet flexible, knowledge-level framework as proposed in this paper will, in our opinion, support the process of knowledge acquisition in a number of ways. First of all, it supports the conceptualization of a domain. Concepts can be specified in a precise manner in terms of sorts, constants, functions and predicates. The notion of a signature can be very usefully exploited for the introduction of appropriate mnemonic names; specifically, sorts are helpful in indicating high-level knowledge types (*e.g.*, data, feature, hypothesis). The axioms in the specification constitute a natural way for expressing factual and control knowledge, without the burden of representational or implementational issues. In addition, it is open to gradual refinement and may thus be of use in high-level prototyping. Second, the introduction of knowledge modules, of module algebra and of the lifting concept supports a, mathematically precise, structuring of a domain, thereby including associated meta-knowledge. A point of interest here is that this opens up the opportunity for using 'conventional' applied mathematics for the purposes of knowledge acquisition and analysis. Finally, as a result of its integration of conceptual and formal aspects, the suggested specification approach supports a clear intermediate documentation of expertise at the knowledge level, on the one hand, and facilitates the transformation from knowledge models to KBS design and implementation, on the other hand. As a whole, the present framework is believed to provide useful guidance to the knowledge engineer's activity.

Acknowledgment We are grateful to Dr. Angi Voß (GMD, Bonn) and Dr. Frank van Harmelen (University of Edinburgh) for their useful comments on earlier versions of this paper.

REFERENCES

H. Akkermans, B. Wielinga, G. Schreiber, and J. Balder. *Towards a Formal Specification of Knowledge Models.* Restricted Distribution Report ECN-89-006, ECN, Petten, The Netherlands, 1989..

J.M. Akkermans, B.J. Wielinga, A.Th. Schreiber, and J.R. Balder. Naar een formele specificatie van kennismodellen. In R.R. Bakker and L. Siklóssy, editors, *Proceedings NAIC-89, Twente*, Academic Service, april 1989.

J.H. Alexander, M.J. Freling, S.J. Shulman, S. Rehfuss, and S.L. Messick. Ontological analysis: an ongoing experiment. In J. Boose and B. Gaines, editors, *Knowledge-Based Systems, Volume 2: Knowledge Acquisition Tools for Expert Systems*, pages 25–37, Academic Press, London, 1988.

J.A. Bergstra, J. Heering, and P. Klint. Module algebra. *Journal of the ACM*, 1989. to appear.

J. Breuker, B. Wielinga, M. van Someren, R. de Hoog, G. Schreiber, P. de Greef, B. Bredeweg, J. Wielemaker, J-P Billault, M. Davoodi, and S. Hayward. *Model Driven Knowledge Acquisition: Interpretation Models.* Esprit Project P1098, Deliverable D1 (task A1), University of Amsterdam and STL Ltd,, 1987.

B. Chandrasekaran. Generic tasks as building blocks for knowledge-based systems: the diagnosis and routine design examples. *Knowledge Engineering Review*, 1988. to appear.

W.J. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985.

P. de Greef and J. Breuker. A case study in structured knowledge acquisition,. In *Proceedings of the 9th IJCAI*, pages 390–392, Los Angeles, 1985.

T. DeMarco. *Controlling Software Projects.* Yourdon Press, New York, 1982.

L. Eshelman, D. Ehret, J. McDermott, and M. Tan. MOLE: a tenacious knowledge acquisition tool. In J.H. Boose and B.R. Gaines, editors, *Knowledge Based Systems, Volume 2: Knowledge Acquisition Tools for Exoert Systems*, pages 95–108, Academic Press, London, 1988.

K. Futatsugi, J. Goguen, J. Jouannaud, and J. Meseguer. Principles of OBJ2. In B.K. Reid, editor, *Proc. Twelfth Symposium on Principles of Programming Languages*, pages 52–66, ACM, 1985.

M.R. Genesereth and N.J. Nilsson. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, California, 1987.

J.A. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logical semantics. In B. Shriver and P. Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477, MIT Press,

Cambridge, Mass., 1987.

S.A. Hayward, B.J. Wielinga, and J.A. Breuker. Structured analysis of knowledge. *International Journal of Man-Machine Studies*, 26:487–498, 1987.

W. Jansweijer. *PDP.* PhD thesis, University of Amsterdam, 1988.

H.B.M. Jonkers. *An Introduction to COLD-K.* Deliverable Esprit Project P432 METEOR/t8/PRLE/8, Philips Research Laboratories, Eindhoven, The Netherlands, July 1988.

J. McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In S. Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–255, Kluwer Academic Publishers, The Netherlands, 1989.

A. Newell. The knowledge level. *Artificial Intelligence*, 1982:87–127, 1982.

G. Schreiber, J. Breuker, B. Bredeweg, and B. Wielinga. Modelling in KBS development. In *Proc. 2th European Knowledge Acquisition Workshop, Bonn*, 1988.

L. Steels. *Components of Expertise.* AI Memo 88-16, AI Lab, Vrije Universiteit Brussel, 1988.

R. Weyhrauch. Prolegomena to a theory of mechanized formal reasoning. *Artificial Intelligence*, 13, 1980.

B.J. Wielinga and B. Bredeweg. Knowledge and expertise in expert systems. In G.C. van der Veer an G. Mulder, editor, *Human-Computer Interaction: Psychonomics Aspects*, pages 290–297, Springer-Verlag, Berlin, 1988.

B. Wielinga and J. Breuker. Interpretation of verbal data for knowledge acquisition. In T. O'Shea, editor, *Advances in Artificial Intelligence*, pages 41–50, ECAI, Elsevier Science publishers, Amsterdam, 1984.

B. Wielinga and J. Breuker. Models of expertise. In *Proceedings ECAI'86*, pages 306–318, 1986.